Conservative Offline Q Learning with Gaussian Processes

Jingyu Liu, Songyan Hou

Introduction

Offline Reinforcement Learning, different from their traditional counterparts, relies solely on previously collected static data to learn a policy without further interaction with the environment. This setup can be tremendously applicable in large-scale problems where querying environments can be expensive or dangerous, like robotics and healthcare. However, it has been shown that on-policy or off-policy algorithms sometimes fail to achieve competitive performance. There are some challenges that Offline RL aims to address:

- How to deal with out of distribution data during deployment?
- How to achieve better performances with the lack of feedbacks from the environment (answering counterfactual problems)?
- How to “stitch” trajectories from the static dataset to achieve “learning” beyond imitation and cloning, especially when the behaviour policies are sub-optimal or even random?

Algorithm Description

To mitigate the overestimation error of the Q function caused by bootstrapping, we penalize the training targets for out-of-distribution (OOD) and sparse state-action pairs in the dataset. Generally, we adopt a Bayesian model that gives an uncertainty estimation for a prediction to penalize the Q values for uncertain state-action pairs in the bellman backup operator.

$$\mathcal{T}_C Q(s,a) := \overbrace{r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}}(Q(s',a')}^{\text{bellman optimality operator}} \underbrace{-\tau \text{Unc}(Q(s',a')))}_{\text{uncertainty penalty}} \tag{1}$$

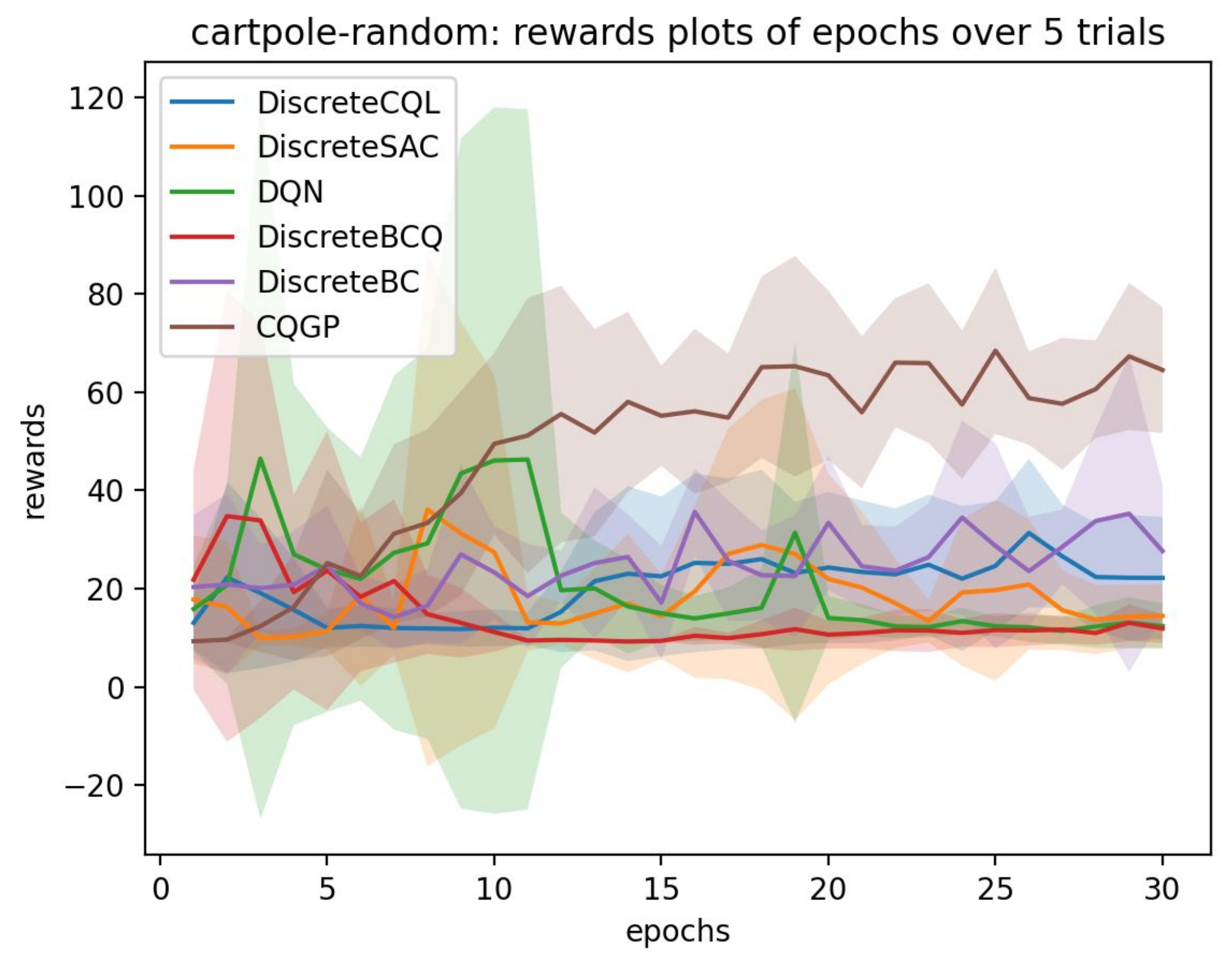
In this project, we chose GPs to model and interpolate the uncertainty of state-action pairs, which can potentially generalize well in cases where the smoothness of the Q function holds. The algorithm is called CQGP.

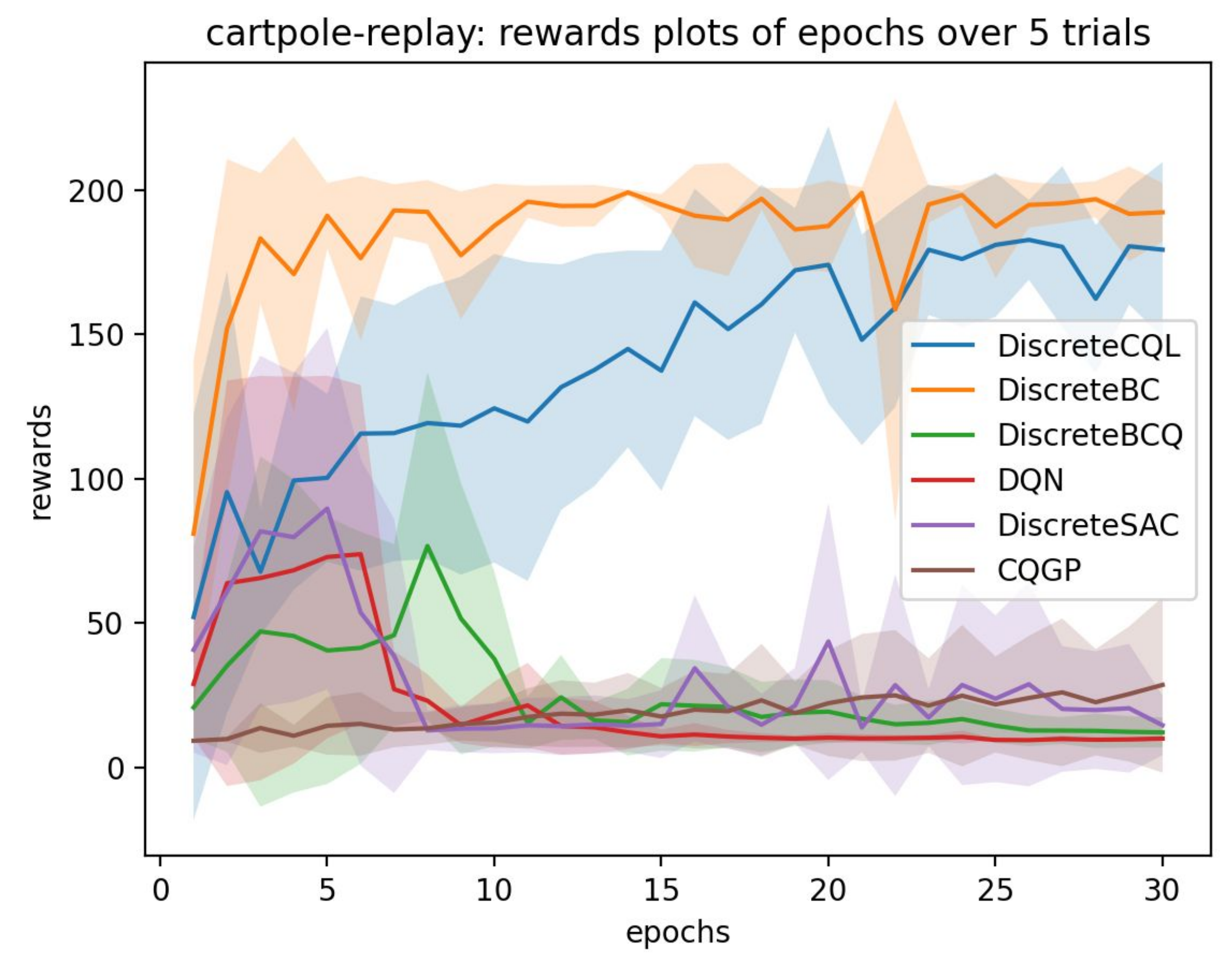
```
Algorithm 1 Conservative Offline Q-Learning with Gaussian Processes
Require: $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}\}_N, \tau, \gamma, \alpha, K$
  $Q \leftarrow$ initialize GP with some prior                ▷ potentially with domain knowledge
  for i from 1 to K do
      $Q' \leftarrow$ initialize GP with some prior            ▷ maybe get some info from $Q$
      $\mathcal{B} \leftarrow \{\}$                                      ▷ empty buffer
      for $\{s, a, r, s'\} \sim \mathcal{D}$ do
          $target \leftarrow r + \gamma \max_{a'}(\mu_Q(s',a') - \tau\sigma_Q^2(s',a'))$   ▷ from equation (1)
          $y = \mu_Q(s,a) + \alpha(target - \mu_Q(s,a))$
          Store pair $(s,a), y \rightarrow \mathcal{B}$
      end for
      Train $Q'$ on $\mathcal{B}$                         ▷ any standard GP training paradigm
      $Q \leftarrow Q'$
  end for
  return $Q$
```

Experiments

The experiment is conducted with the framework D3RLpy where we implemented the proposed algorithm and compare it with CQL, SAC, DQN, BCQ, and behaviour cloning.

Due to computational issues, we chose two datasets collected by policies in the gym environment CartPole. We intentionally restrict the size of the datasets to be small in order to make OOD data more likely to occur during deployment. The figure on the top uses the dataset collected by random policies and the one on the bottom by experts. Both datasets contain very few trajectories (20, 10) because almost all algorithms can solve the problem with sufficient amount of data.





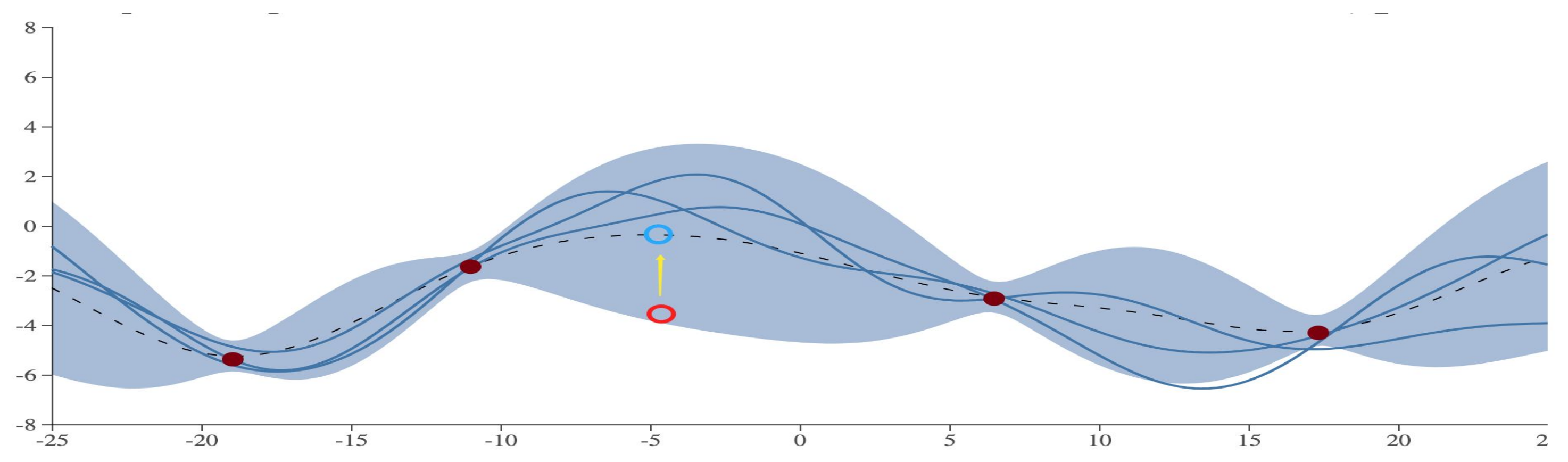


Figure: Illustration of How CQGP Produces a Conservative Estimate for the Q function: the horizontal axis denotes the action space, the vertical axis measures the estimated Q value for a fixed state.

Each algorithm is run over five trials with fixed seeds and the mean is plotted along with the standard deviation. Note that we used the default parameters for all the baselines and did not tune the parameters for CQGP to make the comparison fair (taking into account the sensitivity of hyperparameters).

To make the interpolation smoother and the training more stable, we picked the kernel of the GP to be a rational quadratic with a scaled constant and some white noise, the parameters of which are pre-specified without being optimized during training due to computational burden. The policy induced by the Q function is also regularized by the uncertainty in the similar fashion as the targets. The code is available at https://github.com/Jingyu6/forl_2021

Analysis and Future Works

CQGP outperforms all the baselines in the randomly collected dataset but fails to beat CQL and BC in the dataset collected by experts. We have hypothesized several reasons for such a phenomenon:

- With the expert trajectories, the data density around initial states can be lower than the other case and neural networks may generalize better.
- CQGP is based on standard Q Learning which may not estimate accurately without sufficient amount of visitation, which can be detrimental in the replay dataset which only contains good trajectories.
- The training of CQGP does not optimize the parameters for the GP Q function, which may result in poor performance.

Despite out-performing DQN in the replay dataset, CQGP learns rather slowly (potentially due to the fact that we used the Q learning update). However, different from DQN and SAC which have decreasing performance as training goes on, the improvement is monotonic, implying that it is robust to overestimation of OOD state-action pairs. There are rooms for improvement on the efficiency of the algorithm:

- Can we avoid storing all transitions and make mini-batch updates?
- How to tune or optimize the parameters of the GP Q function?
- Can we use other Bayesian models for the sake of scalability?