CQGP: Robust Conservative Offline Q-Learning with Gaussian Processes

Jingyu Liu
liujin@student.ethz.ch

Songyan Hou
songyan.hou@math.ethz.ch

Abstract

Reinforcement learning (RL) has been shown successful to effectively solve challenging problems with enough feedbacks from the environment. But sufficient interaction with the environment can be expensive or infeasible in certain domains. Offline RL leverages large previously-collected static datasets to learn policies without accessing the environment. However, commonly used off-policy algorithms have been shown to suffer from the overestimation of the state-action values for the out-of-distribution data and the distributional shift between the behaviour policy and the learned policy, especially when training with complex and multi-modal data distributions[1]. In this paper, we propose a robust conservative Q-Learning algorithm with Gaussian Process, which aims to address these limitations by regularizing the estimation of the Q values with the uncertainty of each state-action pairs provided by the Gaussian Process. We introduced a general Bayesian formulation of the algorithm and showed that it can achieve comparable results to the state-of-the-art offline RL algorithms and outperform several offline RL baselines in the classical control environments from OpenAI Gym.

1 Introduction

There have been numerous works that focus on extracting policies with best possible utilities from offline datasets [1]. It has been both observed and proved in the offline setting that the overestimation of Q functions trained from the static transitions can be detrimental to the performance of the policy during deployment [2, 3, 4]. In this project, we aim to design an offline Q learning framework that can produce conservative estimation of the true values and at the same time take advantage of the generalization power of statistical models. Conservative Q Learning with Gaussian Processes (CQGP)[1] uses a Gaussian Process (GP) [5] to model a distributional Q function that can produce state-action value uncertainty based on the occurrences of state-action pairs in the datasets. We introduce a modified bellman backup operator that takes into account both the mean and standard deviations from the GP to produce a conservative target for the Q function. This modified operator is not restricted to Gaussian processes and can be implemented with other Bayesian models such as Bayesian Neural Networks or Ensembles. Our main contributions are summarized as follows:

1. Introduced the modified Bellman operator for estimating conservative Q values in the offline reinforcement learning setting.
2. Implemented the Gaussian Process as the Bayesian model in the modified operator, which results in CQGP.
3. Conducted experiments to verify the effectiveness of CQGP, which outperforms two off-policy RL algorithms by a large margin and achieves comparable or even superior results than some previous offline RL algorithms in OpenAI Gym classical control environments.

[1]The code is available at: `https://github.com/Jingyu6/forl_2021`

Foundations of Reinforcement Learning HS2021, Department of Computer Science, ETH Zürich.

2 Background

2.1 Offline Reinforcement Learning

The offline RL problem can be defined as a *data-driven* formulation of the RL problem. The final goal is equivalent to the one in the online setting:

$$\max_{\pi} J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)}[\sum_{t=0}^{H} \gamma^t r(s_t, a_t)] \tag{1}$$

where $\pi, p_\pi, H, \gamma, r(\cdot,\cdot), s, a$ denote the policy, the trajectories under the policy, the horizon, the discount factor, the reward function, the states and actions in trajectories. However, the agent in the offline setting no longer has the ability to interact with the environment and collect *additional* transitions. Instead, only a *static* dataset of transitions, $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$, is provided and the agent must learn the best policy it can using this dataset. This closely resembles the supervised learning problem statement where $\mathcal{D}$ can be treated as the *training set* but it lacks a proper validation procedure. So the challenge becomes constructing a policy $\pi(a|s)$ that attains the largest possible cumulative reward *when it is actually deployed to interact with the MDP*. Some common solutions include using importance sampling to correct the distribution mismatch, approximate dynamic programming, policy constraints, and uncertainty estimation, all of which have their respective disadvantages. [1] presented a fairly comprehensive overview of these methods.

2.2 Gaussian Processes

We briefly introduce some preliminaries for Gaussian Process (GP) and discuss our approach to use it for approximating Q functions. We focus on the GP regression setting and think of it as defining a distribution over functions and inferencing directly in the function space. Formally, *a GP is a collection of random variables, any finite subset of which has a joint Gaussian distribution*. We can specify a GP completely by its mean function $m(x)$ and covariance function $k(x, x')$:

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)], \\ k(x, x') &= \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \end{aligned} \tag{2}$$

We can assume that the mean function is 0 without loss of generality. Inference in a GP is to predict $f(x^*)$ for a given input x, which involves calculating the posterior distribution:

$$p(f^*|x_1, ..., x_n, y_1, ..., y_n) = GP(\mu', k') \tag{3}$$

where x_{iN} are the features and y_{iN} are the labels in the dataset. Since any finite subset of the random variables are normally distributed, the posterior mean and covariance functions have closed-form solutions:

$$\begin{aligned} \mu'(x) &= \mu(x) + k_{x^*X}(K_{XX} + \sigma^2 I)^{-1}(y - \mu(X)) \\ k'(x, x') &= k(x, x') - k_{x^*X}(K_{XX} + \sigma^2 I)^{-1} k_{x^*X}^T \end{aligned} \tag{4}$$

where K is the covariance matrix, k the covariance vector defined by the covariance function, (X, y) are the feature vectors and labels from the dataset, and σ^2 is the observation noise. Notice that since the covariance function is positive definite, we know that the uncertainty about our prediction will become less as we have more data and its magnitude does not depend on the labels, which is the key property we are taking advantage of when using GP in our algorithm. In the Q learning setting where GP is used to approximate Q values, the feature vectors x can be the concatenation of the state and action pairs (or its embedding produced by some pretrained neural networks), and the label y is the Q values associated with the state-action pairs. Then $p(f(s, a)|\mathcal{D})$ is a predictive distribution about $Q(s, a)$, which is Gaussian with uncertainty represented by the posterior variance (when we have a single state-action pair). For a more in-depth discussion on the subject, please refer to [5].

3 Related Work

Before introducing CQGP, we mention some prior work in offline RL and off-policy evaluation:

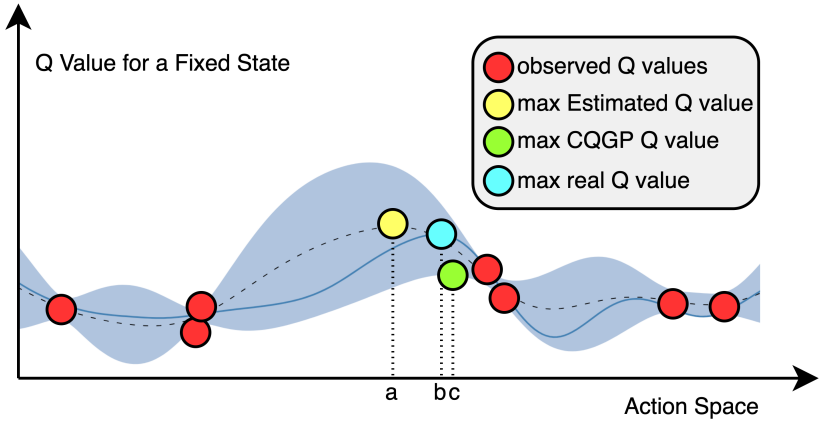


Figure 1: A Toy Visualization of how CQGP Produces Conservative OOD Q Value Estimate: This cartoon describes the Q function for a fixed state s. The horizontal axis denotes the action space and the vertical axis represents the Q value for a specific action at s. The dotted line is the estimated Q function $\tilde{Q}_s$ with the shaded area being the uncertainty (variance) $\tilde{\sigma}_s$. The blue line is the true Q function value Q^*_s for the given state. All the red disks represent the labels in the dataset. As we can see $a = \arg\max \tilde{Q}_s, b = \arg\max Q^*_s, c = \arg\max(\tilde{Q}_s - \tilde{\sigma}_s)$ are the greedy action w.r.t. the estimated Q function, the optimal action w.r.t. the true Q function, and the action chosen by CQGP. In this case, we can see that CQGP selects better action than the greedy action chosen from the point estimate of the Q function.

1. **Conservative Q-Learning (CQL)** in [4], learns conservative (lowerbound) estimates of the Q function, by augmenting the standard bellman error with a regularizer. One variant of CQL, which is of particular interest and is easy to implement, penalizes extreme values (by regularizing the policy to be relatively close to a uniform policy) and maximize values for state-action pairs that exist in the dataset. It outperforms the prior offline reinforcement learning algorithm.
2. **Batch-Constrained deep Q-Learning (BCQ)** in [2], uses a state-conditioned generative model to produce actions with high density in the training data. This generative model is combined with a Q-network, to select the highest valued action which is similar to the data in the batch to overcome extrapolation error in off-policy learning, which implicitly constrains the support of action distribution for a given state.
3. **Bootstrapping error accumulation reduction (BEAR)** in [3] learns a policy to maximize Q-values while penalizing it for deviating away from behaviour policy support. BEAR measure the divergence using kernel MMD and uses a soft-updated ensemble of target Q functions to avoid the overestimation error.

4 Algorithm

In this section, we first start with a general Bayesian formulation of the algorithm, which modifies the Bellman Optimality Operator. Then we reason about choosing the Gaussian Process as the Bayesian model. Finally, we compare our methods to prior works in terms of the difference.

4.1 Bayesian Formulation of the Algorithm

The major issue of using off-policy algorithm for offline dataset is the overestimation of the Q function caused by the boostrapping error and lack of further feedbacks from the environment. Out-of-distribution (OOD) data are extremely detrimental in this case when their estimates are not correct, leading to error exponentially proportional to the length of the horizon. Therefore, the natural idea is to penalize the Q value for OOD state-action pairs so that the policy will be more pessimistic about state-action pairs that are not often seen in the dataset. We propose a general framework that adopts a

Bayesian model to produce an uncertainty estimate for each Q value and penalize the target of OOD state-action pairs with this uncertainty in the Bellman backup operator:

$$\mathcal{T}_C Q(s,a) := \overbrace{r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathcal{A}} (Q(s',a')}^{\text{bellman optimality operator}} \underbrace{-\tau \text{Unc}(Q(s',a')))}_{\text{uncertainty penalty}} \tag{5}$$

where τ controls the level of pessimism. In each iteration, the optimization objective with a fixed evaluation of Q^k is therefore:

$$\min_Q \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}[(Q(s,a) - \mathcal{T}_C Q^k(s,a))^2] \tag{6}$$

The conservative greedy policy induced from the Q function takes a similar form with a different hyperparameter η:

$$\pi(s) = \arg\max_a [Q(s,a) - \eta Unc(Q(s,a))] \tag{7}$$

In this case, the Q function estimate will lowerbound the one produced by the standard Bellman backup operator. Both τ and η, on the one hand, encode our confidence in the generalization power of the Bayesian model as well as how conservative we require the policy to be. On the other hand, they serve slightly different purposes and are often more or less sensitive: τ is usually smaller as it directly affects the convergence of the training procedure whereas the value of η can be freely chosen, even with drastically different value when selecting each new action and can even be state dependent with some domain knowledge. Also if $\tau = 0$, we have CQGP degenerates to behaviour cloning as $\eta \to \infty$ and approaches to the standard Q learning as $\eta \to 0$.

4.2 Gaussian Process Implementation

As mentioned above, any Bayesian model can in principle suffice, and in this work, we chose to use a Gaussian Process. The reason of our choice is that the smoothness of the model can be controlled by the choice of the kernel function so that we would expect reasonable performance when the smoothness of the real Q function holds, which allows us to take advantage of the generalization power of the model. Another motivation behind it is that, unlike several models like ensembles and Bayesian neural networks, a GP models the true *variance* which has statistical meaning and shrinks as more data around the prediction point are observed.

We use a GP to predict the Q value of a state action pair. And we restrict the scope to environments with discrete action space. Hence, all the actions are one-hot encoded and concatenated with the state vector to form the feature vector for the GP. The label is the modified Bellman target described in (5). When we want to make greedy action selection, we make a prediction for each action in a given state, and choose the action base on (7). The overall algorithm is summarized as:

The μ_Q and σ_Q^2 are the posterior mean and variance of the GP at the point (s,a). In each of the iteration, we're fitting a new GP to estimate the current Q value function with a penalty on the inputs that have very low density in the datasets (high variance in the fitted GP). For state-action pairs that appear infinitely many times in the dataset, the variance will approach 0, which recovers the normal bellman operator.

4.3 How Is It Different From Previous Offline RL Algorithms?

Encouraging the learned policy to be close to the behaviour policy is one of the common themes in previous offline RL algorithms. Most of the efforts following this intuition can be divided into two categories: 1) make the learned policy approach the behaviour policy in distribution or 2) make the learned policy have similar support to the behaviour policy in each state. They either blindly penalize OOD state-action pairs, leading to overly pessimistic policy, or require choosing a suitable measure of similarity between policies, which can be domain-specific and calculation heavy. CQGP, however, does not need any specification of such similarity measure nor simply penalizes OOD state-action

Algorithm 1 Conservative Offline Q-Learning with Gaussian Processes

Require: $\mathcal{D} = \{s_t, a_t, r_t, s_{t+1}\}_N$, τ, γ, α, 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
    $\mathcal{B} \leftarrow \{\}$ ▷ empty buffer
    for $\{s, a, r, s'\} \sim \mathcal{D}$ **do**
      $target \leftarrow r + \gamma \max_{a'}(\mu_Q(s', a') - \tau\sigma^2_Q(s', a'))$ ▷ from equation (5)
      $y = \mu_Q(s, a) + \alpha(target - \mu_Q(s, a))$ ▷ standard tabular update for Q function
      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

pairs. It takes into account the statistical confidence about each prediction, taking the full potential of the generalization power of the model. It also decouples training from testing, which gives finer control over how conservative the policy should be in each scenario. Besides, the framework is very easy to implement as it only replaces the Q function approximator with a Bayesian model.

5 Experiments

5.1 Setup

To picture a comprehensive view of the effectiveness of CQGP, we tested it against:

1. Off-policy RL algorithms: Deep Q-Network (DQN) [6] and Soft Actor-Critic (SAC) [7]
2. Offline RL algorithms: Batch-Constrained Q Learning (BCQ), Conservative Q Learning (CQL)
3. Supervised learning algorithms: Behaviour Cloning (BC)

All the algorithms use generic hyperparameters without further tuning from cross-validations. We ran each algorithm in each environment five times with fixed random seeds and plotted the discounted sum of rewards produced by the policy after every epoch, plus and minus one standard deviation from the mean are shaded accordingly.

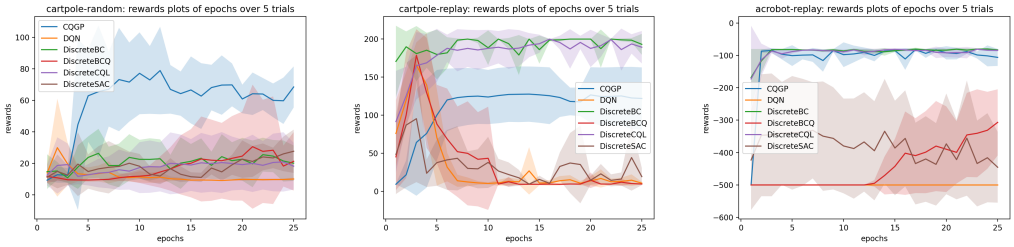
5.1.1 Environments and Datasets

To make comparison fair, We restrict our scope to environments with discrete action space and non pixel inputs so that no further feature embedding is required before feeding the inputs into the GP. Therefore, we tested all the above algorithms in OpenAI Gym classical control environments `CartPole-v0` and `Acrobot-v1`.

We used the offline RL framework from UC Berkeley called d3rlpy[8] to conduct the experiments, which is one of the most popular and standardized offline RL benchmark library. Due to the limited available data (only one environment satisfying our requirement is not deprecated), d3rlpy provides two datasets for `CartPole-v0`, one containing almost random episodes (potentially collected by sub-optimal policies), and one containing episodes from expert policies. We additionally implemented a PPO algorithm [9] to collect expert episodes in `Acrobot-v1`.

5.1.2 Implementation Details

CQGP is implemented using the GaussianProcessRegressor from the Sklearn package [10], and is integrated into the d3rlpy framework by inheriting from the base class algorithm. All other algorithms are implemented with Pytorch[11]. The code with reproducible results is available at `https://github.com/Jingyu6/forl_2021`. We chose the kernel function to be the rational



(a) CartPole-v0 Random (b) CartPole-v0 Replay (c) Acrobot-v1 Replay

Figure 2: The performances of different algorithms on `CartPole-v0` and `Acrobot-v1` environments with random and replay trajectories. Each algorithm is trained using the same of seeds and the mean and standard deviations of the discounted sum of rewards are plotted.

quadratic kernel scaled by a constant with an additive white kernel to capture the potential noises. To alleviate the computational burden, all the hyperparameters of the kernel functions are fixed so that the only calculation in the training process involves matrix inversions and the smoothness property does not change.

5.2 Analysis

We visualized the experiment results shown in 2. Here we listed some of our conclusions:

1. `CartPole-v0` with random trajectories 2a: CQGP outperformed all other methods in terms of the discounted sum of rewards after each epoch and achieved relatively decent variance at the end of training. The average rewards that CQGP scores at the end is higher than the average rewards in the random trajectories (around 22), which indicates that CQGP actually accomplished what is called "stitching" effect, the ability to combine several sub-optimal trajectories and produce trajectories with higher rewards.

2. `CartPole-v0` with replay trajectories 2b: This experiment differs from the first one in that trajectories are collected by expert policy. CQGP outperformed all but CQL and BC. BC is expected to perform well due to the high quality of the trajectories. We hypothesized several reasons for why CQGP is eclipsed by CQL:

   (a) The transition collecting policy for this dataset is not documented. Therefore, it is possible that with the expert trajectories, the data density around initial states can be low and neural networks may generalize better.

   (b) CQGP is based on standard Q learning which may not estimate accurately without sufficient amount of visitation in this environment, which can be detrimental in the replay dataset which only contains good trajectories.

   Despite failing to surpass CQL, the performance of CQGP is increasing monotonically, showing that it does not suffer from the overestimation problem. The same does not hold for BCQ, SAC, and DQN because there are very high peaks in the early epochs and the performances gradually degrade as the training goes on. In this case, we can see that there is little discrepancy between how good CQGP *thinks it does* and how good *it actually does*.

3. `Acrobot-v1` with replay trajectories 2c: CQGP successfully achieved comparable results to BC and CQL, outperforming all other baselines in terms of the rewards and variance.

There are, however, some computational issues that we wish to highlight here. The training process of CGPQ is very slow even in the simple environments with only 2k+ offline transitions with low state and action dimension, which corresponds to at most these many points when calculating the posteriors of the GP. More complex environments like Atari or robotic controls are hard to train due to large amount of data and the pixel-valued inputs, making GP less likely to win over neural networks with convolutional layers.

6 Future Work

The major issue about GP is its computational inefficiency, especially in larger offline datasets with high dimensional input space. Also, commonly used kernels only measure euclidean relationship between data points like absolute difference or squared difference. In lots of domains like image processing, a lower dimensional learned embedding (by neural networks) is usually beneficial for feature extraction and the final performance. Moreover, in the current implementation, fitting a new model is required for each iteration, leading to a very long training time. There are some potential future works that can try to solve the problem:

1. Try other implementations of GP like GPytorch [12].
2. Change the implementation to avoid refitting a GP every iteration.
3. Try other Bayesian models like Bayesian neural networks.

7 Conclusions

In this work, we proposed a new algorithm based on the modified Bellman operator and the Gaussian Process to solve the offline RL problem. We discussed its intuition and conducted experiments to verify its effectiveness compared to several off-policy and offline RL baselines. Some potential drawbacks of the method were mentioned and some future works were pointed out. Finally, we believe that the general framework should be a promising approach upon which other offline RL algorithms can be designed to tackle with the learing problem in the offline RL setting.

References

[1] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems.

[2] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. URL `https://github.com/sfujim/BCQ`.

[3] Aviral Kumar, Justin Fu, George Tucker Google Brain, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. .

[4] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. .

[5] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[8] Michita Imai Takuma Seno. d3rlpy: An offline deep reinforcement library. In *NeurIPS 2021 Offline Reinforcement Learning Workshop*, December 2021.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*,

pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[12] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, 2021.