Enhancing Lunar Lander Performance with Offline-online combined CQL-SAC Approaches

Abstract

In this project, we tackle the classic reinforcement learning challenge of the Lunar Lander environment using a combination of Soft Actor-Critic (SAC) and Conservative Q-Learning (CQL). The objective of the project is to train an agent capable of safely landing a spacecraft on the moon's surface, optimizing fuel efficiency and minimizing risks. By integrating SAC's advantages in continuous action spaces with CQL's strength in managing overestimation bias and improving sample efficiency, we develop a robust model that effectively balances exploration and exploitation. Our methodology includes an initial training phase using SAC to generate high-quality trajectories, followed by a combined online-offline training phase utilizing both SAC and CQL. The experimental results demonstrate that our approach accelerates learning, achieving faster convergence and higher performance compared to traditional methods. This study provides valuable insights into the potential of hybrid learning strategies in complex control tasks and sets the groundwork for future research in applying advanced reinforcement learning techniques to real-world problems.

1 Introduction

1.1 Problem Description

The Lunar Lander environment is based on a classic rocket trajectory optimization problem and presents a classic reinforcement learning challenge where the objective is to control the spacecraft to safely land on the surface of the moon while conserving fuel. In this problem, players are tasked with mastering the adjustments of the spacecraft's thrusters to achieve a smooth landing amidst factors such as gravity and wind. This problem serves as a benchmark to evaluate and compare the performance of various reinforcement learning algorithms in handling continuous action spaces and sparse reward environments. The ultimate goal is to develop algorithms that can efficiently navigate complex and uncertain scenarios, optimizing both safety and resource utilization.

OpenAI's gym deployed an environment to the problem. The agent needs to adjust its actions according to its coordinates, linear velocity and angular velocity in the x, y direction, and to finally land on the landing pad located at $(0, 0)$.

1.2 Objective

In this project, we aim towards training an agent to maximize average score for 100 consecutive episodes in Lunar Lander environment. For reasons elaborated below, we resolve to choose Offline-online combined CQL-SAC to be implemented as the core algorithm for our agent.

1.3 Algorithm Selection

Policy Optimization and Q-Learning are two main model-free RL approaches. While the former is more principled and stable, the latter exploits sampled trajectories more efficiently. Soft Actor-Critic, or SAC, is an interpolation of both approaches. We select the algorithm for our experiment majorly due to the following reasons:

1. SAC is designed to effectively handle environments with high-dimensional, continuous action spaces. Although the Lunar Lander has a discrete action space, SAC's strengths in managing continuous controls can be adapted to discrete actions through various techniques, such as using a *Gumbel-SoftMax* distribution. This capability is useful in finely controlling the lander's thrusters to achieve a smooth landing.

2. It is known for its sample efficiency and stability during training, which is critical in the Lunar Lander setting where the agent must learn to balance and navigate the lander safely to the target location without excessive trial and error, minimizing the risk of destructive behaviors (like crashing).

3. It incorporates an entropy term that encourages exploration by making the policy stochastic. This characteristic ensures that the agent explores a variety of strategies for landing, which can be beneficial in discovering efficient maneuvers that use less fuel or achieve quicker stabilization.

We then address overestimation bias, one of the common problems in Q-learning approaches, with applying Conservative Q-Learning (CQL), which conservatively regularizes the learned Q-values against unseen actions, promoting more reliable and stable learning outcomes. This is particularly important in the Lunar Lander task where overestimating the value of risky actions (like firing engines excessively) can lead to poor performance and catastrophic failures. CQL also effectively reduces extrapolation errors where the model would otherwise make unfounded assumptions about unseen actions.

Furthermore, we resolve to adopt a combined online-offline approach, where we first train a SAC agent to generate trajectories that are stored in a replay buffer, and then train a CQL-SAC agent online combined with this offline data. Again, this method is justifiable from the basis that:

1. by first training an SAC agent, we can ensure that the data collected in the replay buffer is of high quality. SAC, known for its stability and efficiency in continuous action spaces, can explore the environment effectively and generate diverse yet successful trajectories. These trajectories form a rich dataset that contains valuable state-action experiences across a variety of situations.

2. offline data improves sample efficiency because the agent does not have to explore potentially suboptimal actions during its training phase, reducing the amount of experience needed to reach effective performance levels, accelerating the learning process.

2 Methodology

2.1 Environment Setup

In the LunarLander-v2 environment deployed by OpenAI's gym, the Lunar Lander problem is simplified to 8 state spaces and 4 action spaces.

The state spaces can be denote as tuple$(x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, leg_l, leg_r)$, where x and y represents the coordinates of the lander, $\dot{x}$ and $\dot{y}$ represents the velocity of the lander in both directions, θ and $\dot{\theta}$ represents the angle and angular velocity of the lander. leg_l and leg_r are Boolean and indicate whether the lander's legs are in contact with land.

The action spaces include four discrete actions: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

Assuming agent policy π, the score of the nnth episode represents as $s_{n,\pi} = \sum\limits_{t=0}^{T} r(s_t, a_t)$. The our objective function is expressed as $\underset{\pi}{\operatorname{argmax}} \sum\limits_{n=1}^{100} \frac{1}{100} s_{n,\pi}$, where

$$r(s_t, a_t) = \begin{cases} 100, \text{ if "land"} \\ -100, \text{ if "crash"} \\ 10, \; leg_l \; or \; leg_r = 1 \\ -0.5, a_t = \text{"fire main engine"} \\ -0.05, a_t = \text{"fire side engine"} \end{cases} \tag{1}$$

2.2 Policy Gradient

Following stochastic parameterized policy π_θ, we can sample trajectory τ. The aim is to maximize the expected return $J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathbb{E}}[R(\tau)]$. We aim to update θ via gradient descent $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$.

The gradient $\nabla_\theta J(\pi_\theta)$ can be expanded into:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau \sim \pi_\theta}{\mathbb{E}}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau \mid \theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau \mid \theta) R(\tau) \\ &= \int_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) R(\tau) \\ &= \underset{\tau \sim \pi_\theta}{\mathbb{E}}[\nabla_\theta \log P(\tau \mid \theta) R(\tau)] \\ &= \underset{\tau \sim \pi_\theta}{\mathbb{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right]. \end{aligned} \tag{2}$$

2.3 Entropy-Regularized Reinforcement Learning

Q-function tends to dramatically overestimate Q-values, which then leads to the policy breaking because it exploits the errors in the Q-function. To address this issue, we ought to discount the Q-values by some metric.

The entropy H of a random variable $x \sim P$ is defined as:

$$H(P) = \underset{x \sim P}{\mathbb{E}}[-\log P(x)] \tag{3}$$

At each time step t, we give the agent a bonus reward proportional to the entropy of the policy. The Bellman Equation is thus changed to:

$$\begin{aligned} Q^\pi(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathbb{E}}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') + \alpha H(\pi(\cdot \mid s'))\right)\right] \\ &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathbb{E}}\left[R(s,a,s') + \gamma\left(Q^\pi(s',a') - \alpha \log \pi(a' \mid s')\right)\right] \end{aligned} \tag{4}$$

where α is the trade-off coefficient (or temperature). Higher temperature encourages early exploration and prevents the policy from prematurely converging to a bad local optimum.

We can approximate the expectation with samples from the action space:

$$Q^\pi(s,a) \approx r + \gamma\left(Q^\pi(s',\tilde{a}') - \alpha \log \pi(\tilde{a}' \mid s')\right), \quad \tilde{a}' \sim \pi(\cdot \mid s') \tag{5}$$

2.4 Q-Learning Side of SAC

2.4.1 Mean Squared Bellman Error

The Bellman equation describing the optimal action-value function is given by:

$$Q^*(s,a) = \underset{s'\sim P}{\mathbb{E}}\left[r(s,a) + \gamma \max_{a'} Q^*\left(s',a'\right)\right] \tag{6}$$

With sampled trajectories (s,a,r,s',d) stored in replay buffer $\mathcal{D}$, we learn an approximator to $Q^*(s,a)$ with neural network $Q_\phi(s,a)$.

The mean squared Bellman Error (**MSBE**) is computed as:

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbb{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi\left(s',a'\right)\right)\right)^2\right] \tag{7}$$

where $d = 1$ if s' is a terminal state and 0 otherwise.

2.4.2 Target Networks

The optimization target is given by:

$$y\left(r,s',d\right) = r + \gamma(1-d)\max_{a'} Q_\phi\left(s',a'\right) \tag{8}$$

Since we wish to get rid of the parameters ϕ in the target to stabilize the training process, we replace it with the target network ϕ_{targ} which is cached and only updated once per main network update by *Polyak averaging*:

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi. \tag{9}$$

2.4.3 Clipped double-Q

To further suppress Q-values, in SAC we learn *two* Q-functions instead of one, regressing both sets of parameter ϕ with a shared target, calculated with the smaller Q-value of the two:

$$y\left(r,s',d\right) = r + \gamma(1-d)\min_{i=1,2}\left(\max_{a'} Q_{\phi_{i,\text{targ}}}\left(s',a'\right)\right),$$
$$L\left(\phi_i,\mathcal{D}\right) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_i}(s,a) - y\left(r,s',d\right)\right)^2\right].$$

2.5 Policy Learning Side of SAC

Since calculating $\max_a Q_\phi(s,a)$ is expensive, we can approximate it with $\max_a Q(s,a) \approx Q_\phi(s,\mu_{\theta_{\text{targ}}}(s))$, where $\mu_{\theta_{\text{targ}}}$ is the target policy. The objective then becomes to learning a policy that maximizes $Q_\phi(s,a) : \max_\theta \underset{s\sim\mathcal{D}}{\mathbb{E}}\left[Q_\phi\left(s,\mu_\theta(s)\right)\right].$

Here we adopt a squashed state-dependent gaussian policy:

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s)\odot\epsilon\right), \quad \epsilon\sim\mathcal{N}(0,I). \tag{10}$$

Under the context of Entropy-Regularized Reinforcement Learning, we modify the target with:

$$y\left(r,s',d\right) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{tar},j}}\left(s',\tilde{a}'\right) - \alpha\log\pi_\theta\left(\tilde{a}'\mid s'\right)\right), \quad \tilde{a}'\sim\pi_\theta\left(\cdot\mid s'\right) \tag{11}$$

This reparameterization removes the dependence of the expectation on policy parameters:

$$\underset{a\sim\pi_\theta}{\mathbb{E}}\left[Q^{\pi_\theta}(s,a)-\alpha\log\pi_\theta(a\mid s)\right]=\underset{\xi\sim\mathcal{N}}{\mathbb{E}}\left[Q^{\pi_\theta}\left(s,\tilde{a}_\theta(s,\xi)\right)-\alpha\log\pi_\theta\left(\tilde{a}_\theta(s,\xi)\mid s\right)\right] \tag{12}$$

We perform a gradient ascent optimization:

$$\max_\theta \underset{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{N}}}{\mathbb{E}}\left[\min_{j=1,2} Q_{\phi_j}\left(s,\tilde{a}_\theta(s,\xi)\right)-\alpha\log\pi_\theta\left(\tilde{a}_\theta(s,\xi)\mid s\right)\right], \tag{13}$$

2.6 CQL-SAC

Since we're trying to do offline-online combined updates for performance improvement, we need to tackle with the offline reinforcement learning problem with generated samples. From prior works regarding offline RL [6][7], OOD actions and function approximation errors will pose problems for Q function estimation. Therefore, we adopt conservative Q-learning method proposed by prior work [8] to address this issue.

2.6.1 Conservative Off-Policy Evaluation

We aim to estimate the value $V^\pi(s)$ of a target policy π given access to a dataset $\mathcal{D}^\beta$ generated by pretrained SAC behavioral policy $\pi_\beta(a|s)$. Because we are interested in preventing overestimation of the policy value, we learn a conservative, lower-bound Q-function by additionally minimizing Q-values alongside a standard Bellman error objective. Our choice of penalty is to minimize the expected Q-value under a particular distribution of state-action pairs $\mu(s,a)$. We can define a iterative optimization for training the Q-function:

$$\hat{Q}^{k+1}\leftarrow \text{argmin}_Q \alpha\mathbb{E}_{s\sim\mathcal{D}^\beta,a\sim\mu(a|s)}[Q(s,a)]+\frac{1}{2}\mathbb{E}_{s,a\sim\mathcal{D}^\beta}[(Q(s,a)-\hat{\mathcal{B}}^\pi\hat{Q}^k(s,a))^2] \tag{14}$$

where $\hat{\mathcal{B}}^\pi$ is the Bellman operator and α is the tradeoff factor. The optimality for this update as: $\hat{Q}^\pi=\lim_{k\to\infty}\hat{Q}^k$ and we can show it lower-bounds Q^π for all state-action pairs (s,a). We can further tighten this bound if we are only interested in estimating $V^\pi(s)$. In this case, we can improve our iterative process as:

$$\begin{aligned}\hat{Q}^{k+1}\leftarrow \text{argmin}_Q \alpha(\mathbb{E}_{s\sim\mathcal{D}^\beta,a\sim\mu(a|s)}[Q(s,a)]-\mathbb{E}_{s\sim\mathcal{D}^\beta,a\sim\pi^\beta(a|s)}[Q(s,a)])\\ +\frac{1}{2}\mathbb{E}_{s,a\sim\mathcal{D}^\beta}[(Q(s,a)-\hat{\mathcal{B}}^\pi\hat{Q}^k(s,a))^2]\end{aligned} \tag{15}$$

By adding a Q-maximizing term, although it may not be true for $\hat{Q}^\pi$ being the point-wise lower-bound for Q^π, we still have $\mathbb{E}_{\pi(a|s)}[\hat{Q}^\pi(s,a)]\le V^\pi(s)$ when $\mu(a|s)=\pi(a|s)$. For detailed theoretical analysis, we will refer to prior work [8].

2.6.2 Conservative Q-Learning for Offline RL

We now adopt a general approach for offline policy learning, which we refer to as conservative Q-learning (CQL). This algorithm was first presented by prior work [8]. We denote $CQL(\mathcal{R})$ as a CQL algorithm with a particular choice of regularizer $\mathcal{R}(\mu)$. We can formulate the optimization problem in a min-max fashion:

$$\begin{aligned}\text{min}_Q\text{max}_\mu \alpha(\mathbb{E}_{s\sim\mathcal{D}^\beta,a\sim\mu(a|s)}[Q(s,a)]-\mathbb{E}_{s\sim\mathcal{D}^\beta,a\sim\pi^\beta(a|s)}[Q(s,a)])\\ +\frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}^\beta}[(Q(s,a)-\hat{\mathcal{B}}^{\pi_k}\hat{Q}^k(s,a))^2]+\mathcal{R}(\mu)\end{aligned} \tag{16}$$

Since we're utilizing CQL-SAC, we will chose the regularizer as the entropy H, making it $CQL(H)$. In this case, the optimization problem will be reduced as:

$$\min_Q \Big\{ \alpha \mathbb{E}_{s\sim\mathcal{D}^\beta}(\log \sum_a \exp(Q(s,a)) - \mathbb{E}_{s\sim\mathcal{D}^\beta, a\sim\pi^\beta(a|s)}[Q(s,a)]) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}^\beta}[(Q(s,a) - \hat{\mathcal{B}}^{\pi_k}\hat{Q}^k(s,a))^2] \Big\} \tag{17}$$

More specifically, we let the regularizer $\mathcal{R}(\mu) = -D_{KL}(\mu, \rho)$, where $\rho(a|s)$ is a prior distribution. We can then derive $\mu(a|s) \propto \rho(a|s)\exp(Q(s,a))$. We take the prior distribution as a uniform distribution $\rho = \text{Unif}(a)$, making the regularizer as the entropy H. In this way, we can retrieve the optimization target above. For detailed derivations and theoretical analysis we refer to [8].

2.7 Model Architecture

Architecture for SAC and its CQL-modified version is illustrated as follows[1]:

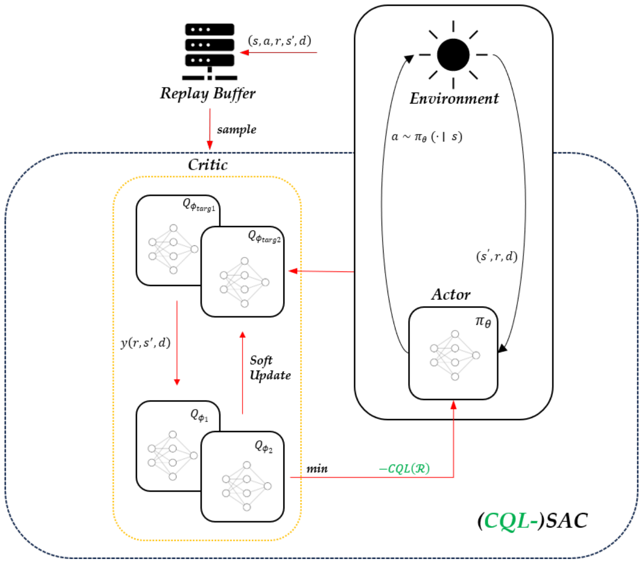


Figure 1: (CQL-)SAC Architecture

The overall pipeline is visualized as figure 2:

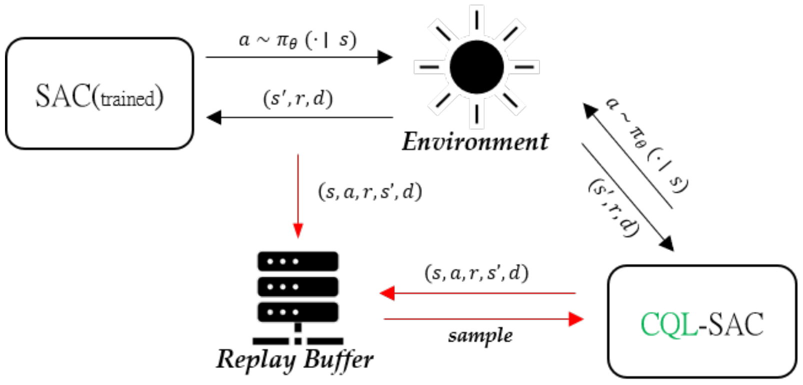


Figure 2: CQL-SAC Pipeline

2.8 Hyperparameters

The hyperparameters commonly used for the CQL-SAC algorithm and their rationale behind selection are as follows:

- *Batch size*: The batch size determines the number of experiences sampled from the replay buffer for each training iteration. Our selected value is 256.
- *Episodes:* The number of episodes defines the total number of episodes the agent will run during training. Our selected value is 1000.
- *Buffer size:* The buffer size refers to the max capacity of the replay buffer, which stores past experiences for training the agent. Our selected value is 100000.
- *Learning rate*: The learning rate controls the step size of the gradient descent optimization process. It influences the speed and quality of learning. Our selected value is $5e$-4.
- *Discount factor (γ)*: The discount factor determines the importance of future rewards in the agent's decision-making process. Our selected value is 0.99.
- *Soft update coefficient* (τ): By gradually blending the parameters of the target networks with those of the current networks, τ determines the rate at which the target networks are updated. Our selected value is 0.01.

3 Results

3.1 Training Performance

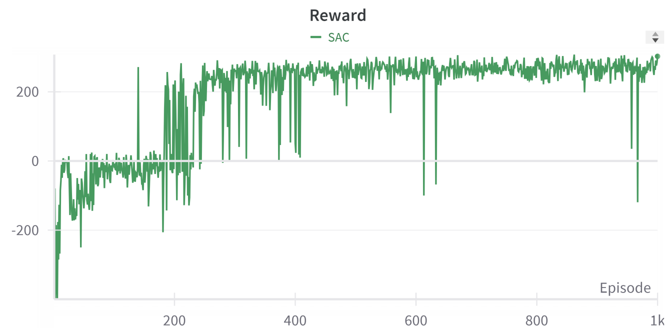


Figure 3: SAC Reward

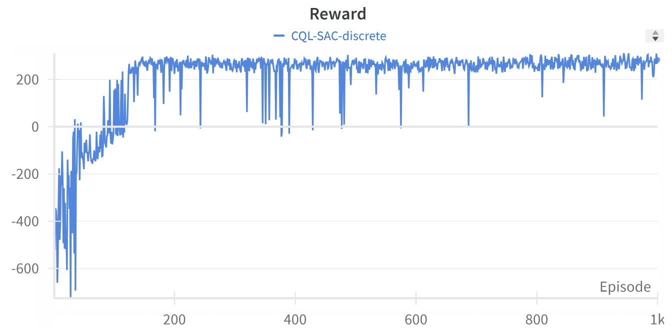


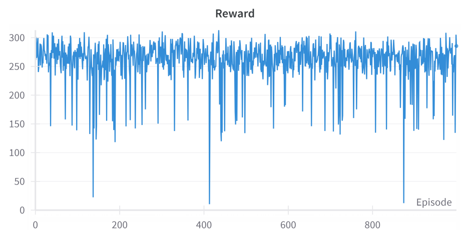
Figure 4: CQL-SAC Reward

Figure 3 and 4 visualizes the rewards at each training epoch for both SAC and CQL-SAC training procedures. We can clearly observe the accelerating effects of the offline dataset generated by the former agent on the latter, in that the CQL-SAC agent reaches reward saturation and stabilizes at around 200 episodes, whereas in comparison it takes the SAC agent approximately 400 episodes to do so. The CQL-SAC also obtains higher average and maximal rewards than the naive SAC agent, however it is not evident due to the relatively low dimensions of the lunar lander environment.

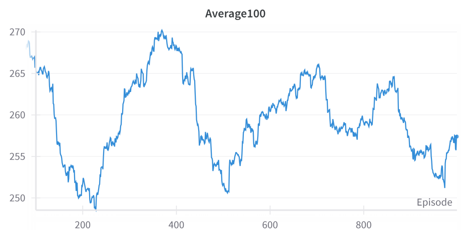
It is also worth noting the CQL-SAC agent continues to produce anomalies after convergence more frequently than the SAC agent, due to the fact that it regularizes the Q-function and prevents over-fitting towards overestimated Q-values.

3.2 Evaluation

After the entire training process is completed, we ran 1000 consecutive episodes with the trained agent and concluded that all sliding windows of 100 consecutive episodes have an average reward > 245, which is generally satisfactory despite a few anomalies which are inevitable.



[a]



[b]

Figure 5: Evaluation Results

3.3 Hyperparameter Sensitivity

After an SAC agent capable of generating high-quality trajectories is trained, it will be utilized to sample episodes that would be dumped into the replay buffer of the CQL-SAC agent, which would take up a certain percentage ρ in its training data. This section will mainly emphasize on the influence of that proportion on the offline-online CQL-SAC agent's performance. For a fixed amount of online data sampled (200 episodes), we take six discrete offline dataset sizes which corresponds to $\rho \in [20\%, 75\%]$, and plot the following figures to support our analysis.

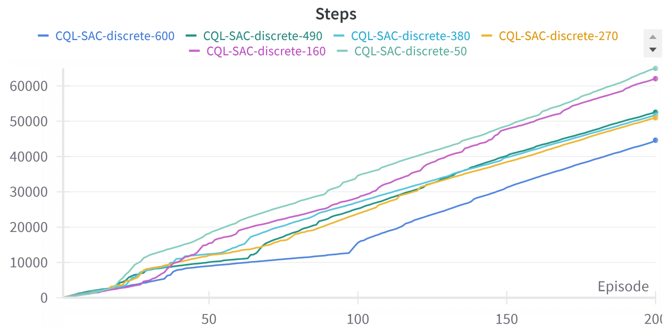
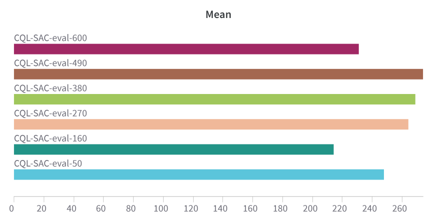


Figure 6: CQL-SAC Steps for Different ρ

Figure 6 illustrates the CQL-SAC agent's total action steps during its training process. It can be observed that higher offline portions leads to reduced action steps for the same amount of episodes, suggesting more efficient and optimized strategies.

Figure 7 depicts the mean and standard deviations of the initial 100 episodes sampled by the trained CQL-SAC agent. It shows that in general, higher ρ corresponds to higher average rewards and lower variance, with the anomalies potentially due to choice of random seeds.

[a] 

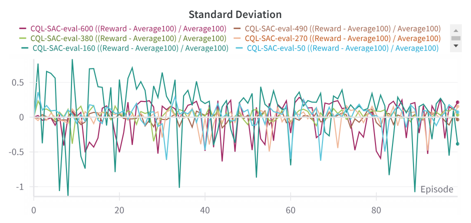
[b] 

Figure 7: Evaluation Reward features for Different ρ

4 Discussion

4.1 Algorithm Analysis

CQL-SAC exhibits promising results in tackling the Lunar Lander problem by effectively improving sample efficiency, striking a balance between exploration and exploitation, and reducing overestimation bias. These strengths contribute to its competitive performance in maximizing return in complex continuous control tasks.

However, the algorithm also has some weaknesses that deserve attention. Like many deep reinforcement learning algorithms, CQL-SAC's performance can be sensitive to hyperparameter tuning. Finding the right combination of hyperparameters for optimal performance can be a challenging and time-consuming process. This part of the work has been explored by the previous researches, so it did not cause us too many difficulties. Additionally, the use of multiple Q-function ensembles and the conservative updates in CQL can increase the computational complexity of the algorithm, potentially requiring more computational resources and training time.

4.2 Challenges and Solutions

We did not encounter any major difficulties when conducting this project. However, due to the inherent drawbacks of SAC and CQL methods, we noticed several schemes that could further optimize our algorithm. For example, when the state visitation distribution of offline data is narrow, the agent is more likely to have a distribution shift, and if the environment is explored in advance, finding high-quality samples close to the strategy can reduce such shift errors. In addition, since the Q function cannot provide an accurate estimate for the online sample outside such distributions, using multiple Q functions to pessimistically evaluate the current strategy can prevent over-optimism about unfamiliar actions in the new state during the initial training phase.

5 Conclusion and Future Work

5.1 Summary of Findings

In summary, we implemented the CQL-SAC algorithm to solve the Lunar Lander continuous control task, as well as explored its strengths and weaknesses. The experiment results show that the CQL-SAC algorithm, with its conservative updates and trade-off between exploration and exploitation, is quite effective in solving complex continuous control tasks. Yet, its performance can be sensitive to hyperparameter tuning and may require more computational resources due to increased complexity.

5.2 Future Improvements

Since CQL and SAC algorithms are all model-free methods, it may pose generalization problem to the agent. Therefore, it may be helpful to adopt model-based algorithms to improve the generalizability. By predicting the forward dynamics $p(s_{t+1}|s_t)$, the model can retrieve the information of the environment dynamics instead of only blindly predict the action. We also may include balanced buffer and ensembles for Q estimation for further improvement.

6 Reference

[1] Spinning Up in Deep Reinforcement Learning, Achiam, Joshua, (2018).

[2] Haarnoja, Tuomas, et al. "Soft actor-critic algorithms and applications."*arXiv preprint arXiv:1812.05905*(2018).

[3] Kumar, Aviral, et al. "Conservative q-learning for offline reinforcement learning."*Advances in Neural Information Processing Systems*33 (2020): 1179-1191.

[4] Zhang, Shangtong, and Richard S. Sutton. "A deeper look at experience replay."*arXiv preprint arXiv:1712.01275*(2017).

[5] Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods."*International conference on machine learning.* PMLR, 2018.

[6] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643, 2020.

[7] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In Advances in Neural Information Processing Systems, pages 11761–11771, 2019.

[8]Aviral Kumar, Aurick Zhou, George Tucker and Sergey Levine. Conservative q-learning for offline reinforcement learning. In Advances in Neural Information Processing Systems, 2020.

7 Appendices

7.1 Pseudocode

Algorithm 1: Soft Actor-Critic

```
input  : $\theta_1, \theta_2, \phi$                                         // Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$                         // Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$                                               // Initialize an empty replay pool
for each iteration do
    for each environment step do
        $a_t \sim \pi_\phi(a_t|s_t)$                                // Sample action from the policy
        $s_{t+1} \sim p(s_{t+1}|s_t, s_t)$                      // Sample transition from the environment
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$   // Store the transition in the replay pool
    end
    for each gradient step do
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J(\theta_i)$ for $i \in \{1, 2\}$     // Update the Q-function parameters
        $\phi \leftarrow \phi - \lambda \hat{\nabla}_\phi J_(\phi)$                            // Update policy weights
        $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$                             // Adjust temperature
        $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$           // Update target network weights
    end
end
output : $\theta_1, \theta_2, \phi$                                       // Optimized parameters
```

Algorithm 2: Conservative Q-Learning

```
Initialize Q-function, $Q_\theta$, and optionally a policy, $\pi_\phi$.
for step t in {1, . . . , N} do
    Train the Q-function using $G_Q$ gradient steps on objective from Equation 17
    $\theta_t := \theta_{t-1} - \eta_Q \nabla_\theta \mathrm{CQL}(\mathcal{R})(\theta)$ (Use $^*$ for Q-learning, $^{\phi_t}$ for actor-critic)
    (only with actor-critic) Improve policy $\pi_\phi$ via $G_\pi$ gradient steps on $\phi$ with SAC-style entropy
      regularization:
    $\phi_t := \phi_{t-1} + \eta_\pi \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\phi(\cdot|s)}[Q_\theta(s, a) - \log \pi_\phi(a|s)]$
end
```