DCE: Offline Reinforcement Learning With Double Conservative Estimation

Anonymous authors
Paper under double-blind review

Abstract

Offline Reinforcement Learning aims to solve the application challenges of traditional reinforcement learning. Offline reinforcement learning relies on previously-collected datasets to train agents without any interaction with environments. In order to address the overestimation of OOD (out-of-distribution) actions, conservative estimation tends to assign a low value to all inputs. Previous conservative estimation methods usually found it difficult to avoid the impact of OOD actions on Q-value estimates. In addition, these algorithms usually sacrifice some computational efficiency for the purpose of conservative estimation. In this paper, we propose a simple conservative estimation method – double conservative estimation (DCE), which is based on two conservative estimation methods to constraint policy. Our algorithm introduces V-function to avoid the error of estimating in-distribution action while implicitly achieving conservative estimation. In addition, our algorithm introduces a controllable penalty term to adjust the degree of conservative in training. We theoretically show how this method influences the estimation of OOD actions and in-distribution actions. Our experiment shows that DCE achieves state-of-the-art performance on D4RL. We observed that both two conservative estimation methods impact the estimation of all state-action.

1 Introduction

Reinforcement Learning has made significant progress in recent years. However, in practice, the application of reinforcement learning is mainly in games that can easily interact with the environment. Although off-policy reinforcement learning uses previously-collected datasets when it trains agents, in practice, off-policy does not perform well without any interaction. In order to apply reinforcement learning in those environments which are hard to interact with (e.g. autonomous driving, medical care), offline reinforcement learning has attracted some attention. Offline reinforcement learning aims to train an agent by previous-collected, static datasets without further interaction and wishes a good performance of this agent, which can be competitive with those online algorithm's agents.

The main challenge for offline reinforcement learning is that critics tend to overestimate the value of out-of-distribution state action, which is introduced by the distribution shift. Another problem of offline reinforcement learning is that it cannot improve exploration due to the absence of environmental interaction. Unfortunately, this problem cannot be solved because of the nature of the model-free offline reinforcement learning algorithm (Levine et al. (2020)). Therefore, the offline reinforcement learning algorithm mainly addresses the overestimation caused by distribution shifts. There are many methods to solve the overestimation caused by distribution shifts. We can divide offline RL algorithm into the following categories: Policy constraint (e.g. BCQ Fujimoto et al. (2019), BEAR Kumar et al. (2019)), Conservative estimation (e.g. CQL Kumar et al. (2020)), Uncertainty estimation (e.g. PBRL Bai et al. (2022)), Using DICE (e.g. AlgaeDICE Nachum et al. (2019)), Using V-function (e.g. IQL Kostrikov et al. (2021b)), Other (e.g. DR3 Kumar et al. (2021)), etc.

In those algorithms, IQL solves a problem which is caused by the Bellman equation: $\mathcal{B}^{*}Q(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\mathbf{s}'\sim P(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[Q\left(\mathbf{s}',\mathbf{a}'\right)\right]$ (a' comes from the policy). IQL learn a V-value of state by expectile regression to control the estimation of state action. By using advantage-weighted regression, the policy of IQL can cover the optimal in-distribution action. However, IQL is too "conservative" to discover the optimal action, which may be near the in-distribution action. On the other hand, previous conservative estimation methods need to lose some computational efficiency to achieve the purpose of conservative estimation.

Our work proposes a method to explore the OOD action to get a better policy appropriately while keeping the Q-values of in-distribution actions relatively accurate. We found that the V-function has a conservative estimation effect, but it is not enough (see Appendix E). Therefore, we need additional conservative estimation to constrain the exploration of OOD action. Our algorithm uses a simple conservative estimation algorithm without requiring pre-training like Fisher-BRC. We use V-function and an additional penalty term to achieve conservative estimation. In addition, our algorithm uses a controllable parameter to change the degree of conservation in training. Our algorithm aims to achieve limited exploration of OOD action by conservative estimation while having little influence on the state-action of the dataset. Our algorithm does not require additional pre-training and maintains higher computational efficiency. In addition, our algorithm is only modified a little for the standard Actor-Critic algorithm. Our main contributions are as follows:

- We propose that using V-function and an additional term limit the exploration of policy to close to the distribution of critics and alleviate the impact of OOD action on in-distribution action. In addition, our experiment shows the influence of the V-function.
- We help V-function to achieve conservative estimation by a simple penalty term, which is the Q-value of action sampled from the current policy. By this method, we do not need to consider in-dataset actions or actions with low Q-value appearing in the penalty term, as in other conservative estimation methods.
- We theoretically demonstrate how this method influences the estimation of OOD actions and in-distribution actions. In addition, we did experiments to show the effect of our algorithm in practice.
- We experimentally demonstrate that our algorithm can achieve state-of-the-art performance on D4RL.

2 RELATED WORK

In recent years, many offline reinforcement learning algorithms have been studied to apply reinforcement learning to more practical fields other than games. We classify those by the method of their algorithm as follows:

- **Policy constraint:** One method to achieve offline RL is policy constraint, which contains implicit and explicit constraints. Explicit constraint direct constrains the agent's policy into behaviour policy which is used to collect datasets, including BCQ (Fujimoto et al. (2019)), DAC-MDP (Shrestha et al. (2020)), BRAC (Wu et al. (2019)), EMaQ (Ghasemipour et al. (2020)), etc. Implicit constraint use implicit method, like f-divergence, to constrain policy close to behaviour policy, including BEAR (Kumar et al. (2019)), ABM (Siegel et al. (2020)), AWAC (Nair et al. (2022)), AWR (Peng et al. (2021)), CRR (Wang et al. (2020)), etc.
- **Conservative estimation:** This method avoids policy selecting OOD action by giving a lower value of OOD state-action because overestimation and irremediable value cause policy extrapolation errors. It include: CQL (Kumar et al. (2020)), Fisher-BRC (Kostrikov et al. (2021a)), CDC(Fakoor et al. (2021)) etc.
- **Uncertainty estimation:** This is another method to avoid overestimation by giving the uncertainty of estimation of the critic or model. The algorithm gives an uncertainty of estimation, and if it exceeds the threshold, it will take specific measures. it include: MOReL (Kidambi et al. (2020)), PBRL (Bai et al. (2022)), UWAC (Wu et al. (2021)), etc.
- **Using V-function:** Another method is using the V-function of the state to avoid overestimating action because V-value does not need action. It also constrains policy, but we separate it from policy constraint because it uses a different Bellman equation. These methods usually use advantage-weighted behavioural cloning to prevent extrapolation. It include: IQL (Kostrikov et al. (2021b)), VEM (Ma et al. (2021)), etc.
- **Using DICE:** The above algorithm does not directly solve the distribution shift problem. DICE solve this problem by calculating discounted stationary distribution rations. It in-

clude: AlgaeDICE (Nachum et al. (2019)), OPtiDICE (Lee et al. (2021)), RepB-SDE (Lee et al. (2020)).

- **Other:** There are some algorithms which don't fall into the above category: DR3 (Kumar et al. (2021)), OPAL (Ajay et al. (2020)), O-RAAC (Urpí et al. (2021)), etc.

Our work focuses on the conservative estimation algorithm and v-function to get good performance while maintaining the algorithm's efficiency and conciseness.

In our work, we try to solve the valuation error of conservative estimation introduced by the Bellman equation. Suppose the constraint of policy use advantage-weighted behavioural cloning; the agent may miss optimal action near the existing policy distribution. However, using V-value without any policy constraint will get a weak performance because of OOD action. Therefore, we need another method to constrain the policy, which avoids the valuation error of conservative estimation and gives a conservative estimation for OOD action. In addition, we control the degree of conservation by updating the parameter in training. Our algorithm includes expecitle regression (IQL, VEM) and conservative estimation. The effect of expecitle regression has been demonstrated by Kostrikov et al. (2021b) and Ma et al. (2021). We simply introduce the expecitle regression content and show its influence on our algorithm. Section 4 mainly shows our penalty term and analyses what it does for our algorithm.

3 PRELIMINARIES

The goal of Reinforcement Learning is to train a policy to maximize the expected cumulative reward, which experts usually set. Many RL algorithms are considered in the Markov decision process (MDP), which contains a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma).\mathcal{S}, \mathcal{A}$ represent state spaces of environment and action spaces of policy, $T\left(\mathbf{s}^{\prime} \mid \mathbf{s}, \mathbf{a}\right)$ is environment dynamic and tell us the transition probability of state s' when policy use action a in state s. $r(\mathbf{s}, \mathbf{a})$ represents reward function, and $\gamma \in (0, 1)$ is the discount factor. $\pi_\beta(\mathbf{a} \mid \mathbf{s})$ represents behavior policy which use to collect dataset D, $d^{\pi_\beta}(\mathbf{s})$ is discount marginal state-distribution of $\pi_\beta(\mathbf{a} \mid \mathbf{s})$. The goal of the policy is to maximize returns:

$$\pi^* = \underset{\pi}{\arg\max} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t\right) \mid s_0 \sim p_0(\cdot), a_t \sim \pi\left(\cdot \mid s_t\right), s_{t+1} \sim p\left(\cdot \mid s_t, a_t\right) \right] \tag{1}$$

The off-policy algorithm achieves the above objectives by utilising the state-action value function (Q-function) Q(s, a).

Offline reinforcement learning. In contrast to the online RL algorithm, offline RL uses previously static collected data training policy without further environment interaction. Recently many model-free offline RL algorithms train critics by minimizing temporal difference error and its loss:

$$L_{TD}(\theta) = \mathbb{E}_{(s,a,s^{\prime}) \sim \mathcal{D}} \left[\left(r(s, a) + \gamma \max_{a^{\prime}} Q_{\hat{\theta}}\left(s^{\prime}, a^{\prime}\right) - Q_\theta(s, a) \right)^2 \right] \tag{2}$$

where D is the dataset, $a^{\prime}$ represents the action which is sample from policy, $Q_\theta(s, a)$ is a parameterized Q-function, $Q_{\hat{\theta}}(s^{\prime}, a^{\prime})$ is target network (usually use soft parameters update), $s^{\prime}$ is next state and $a^{\prime}$ represents action which sample from π. As we can see, if $a^{\prime}$ is an OOD action, Q-function does not estimate it correctly, and TD loss will exacerbate this error. Overestimation caused by accumulative error will lead to policy extrapolation, making it difficult for performance to converge to a good level. So, many offline RL algorithms add a penalty term to solve the problem that Q-function overestimates OOD state-action.

4 DOUBLE CONSERVATIVE ESTIMATES

In this section, we will show our work, which aims to get a relatively accurate in-dataset action estimation and reduce the overestimation value of OOD state-action in training. Therefore, our algorithm must penalise the overestimation of OOD action while avoiding the influence of OOD action on the in-distribution action.

First, for the traditional conservative estimation offline RL algorithm, the critic uses the following target loss function:

$$L_{TD}(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\left(r(s,a) + \gamma \max_{a'} Q_{\hat{\theta}}\left(s', a'\right) - Q_\theta(s,a)\right)^2\right] + penaltyterm \tag{3}$$

a' is a sample from the policy. If this action is an OOD action, Q-function will give a wrong Q-value which is difficult to correct because of the nature of offline RL. In addition, the penalty term must be carefully designed to avoid poor performance introduced by an inappropriate conservative estimation penalty term design.

Unlike the traditional reinforcement learning algorithm that uses conservative estimation, our algorithm can achieve the above goal without introducing additional agent training. Specifically, our algorithm uses the V-function mentioned above to train Q-function and add a penalty term to control the degree of conservatism estimation. Our algorithm reduces the Q-value of the OOD state-action and ensures that the values of the in-distribution state-action will not introduce additional errors.

4.1 Learning Q-function with V-funcion

Our work use SARSA-style objective to estimate in-dataset state-action, which has been considered in prior Offline Reinforcement Learning (Kostrikov et al. (2021b), Ma et al. (2021), Brandfonbrener et al. (2021), Gulcehre et al. (2021)) to achieve the first goal. We train the V-function use the following loss:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau\left(Q_{\hat{\theta}}(s,a) - V_\psi(s)\right)\right] \tag{4}$$

where $L_2^\tau(u) = |\tau - 1(u < 0)|u^2$. As a result, this equation balances the weights of different values of Q concerning values of V. The corresponding loss of Q-function is as follows:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2] \tag{5}$$

The advantage of the above equation is that this equation can eliminate the relationship between the action of the current state and the action of the next state, which can avoid the problem of the bellman equation. In addition, using V-function can appropriately alleviate the problem of overestimation, which we will show experimentally. Furthermore, we see this as a conservation method.

The critic with V-function can cover the in-dataset optimal policy distribution. However, only having an accurate in-dataset action estimation cannot effectively relieve the overestimation of OOD action, so we need to reduce the estimation of OOD action while sustaining relatively accurate in-dataset action estimation.

4.2 OOD Penalty Term

The idea of our algorithm is that for the Q-value of OOD state-action, we do not need to get its exact conservative value but only need its value lower than the value of optimal action strategies to exist in the dataset. For values of OOD state-action that are lower than the values of in-distribution action, we do not care about the change of these. For values of OOD state-action that may be higher, caused by overestimation, than the values of in-dataset action, we hope to bring it down to a lower level. At the same time, the agent can also select better action. Notably, the discussion about OOD actions in CDC differs from ours. CDC only consider the OOD action, which has a higher Q-value than the in-dataset action. CDC reduce the Q-value of these OOD actions and increases the value of in-dataset action. For OOD actions which have a lower Q-value than in-dataset action, CDC doesn't handle it.

We propose a simple penalty term to achieve the above aim. Our work uses Q-value off OOD action (sampled by current policy) as a penalty term to reduce overestimation. At last, Q-function's objective loss is as follows:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\left(r(s,a) + \gamma V_\psi\left(s'\right) - Q_\theta(s,a)\right)^2\right] + \beta \times \mathbb{E}_{s\sim\mathcal{D},a'\sim\pi_\theta}[Q_\theta(s,a')] \tag{6}$$

In this objective, $Q_{\hat{\theta}}(s,a')$ represents Q-value of state-action (s,a') which a' sample from current policy. In this penalty term, our work uses the current policy as an OOD action sampler rather than the randomly initialized policy because low-quality action will push Q-function in the wrong direction. In the experiment, we use the current policy to generate the action of the current state, taking its Q value as the penalty term. We will analyze the influence of this penalty term and give a lower-bound theory.

4.3 Analysis

To facilitate analysis, we set the parameter β to a fixed value, which follows the train regularly. In practice, however, a variable β may have a better performance in some datasets because of the impact of conservative estimation on policy. Furthermore, for variable β, the result of the Q-value and V-value are the same as those of constant β.

First, we fit the V-value to the current best to consider the influence of the Q-value. Then, we consider in-distribution and OOD action for the Q-value of different state-action. The penalty term penalizes the overestimation of OOD action to make it as small as possible. For the case where all state-action pairs are in distribution, we take the derivative with respect to Equation 5:

$$\nabla_\theta L_Q(\theta) = \nabla_\theta Q(s,a) \times ((r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a)) + \beta \times \nabla_\theta Q(s,a) \tag{7}$$

Through this equation, we get the Q-value of in-distribution state-action that minimizes objective loss. As a result, for each iteration, we have the following equation:

$$Q_k(s,a) = r + \gamma V_k(s') - \beta \tag{8}$$

Then, we fit the Q-value to the current best to consider the influence of the V-value. Equation 4 shows that V-function is only affected by the Q-value of in-distribution state-actions, so we can get a relatively accurate V-value estimation of state.

Given the above result $Q_k(s,a) = r + \gamma V_k(s') - \beta$, we substitute it into the equation of V-function $V(s) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q(s,a))$. As a result, we can get follow equation:

$$\begin{aligned} V_{k+1}(s) &= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_k(s,a)) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_k(s') - \beta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_k^*(s') - \frac{1-\gamma^k}{1-\gamma}\beta) \\ &= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_k^*(s,a) - \frac{1-\gamma^k}{1-\gamma}\beta) = V_{k+1}^*(s) - \frac{1-\gamma^k}{1-\gamma}\beta \end{aligned} \tag{9}$$

$Q_k^*(s,a)$ represents Q-value estimated by the original Q-function in the previous iteration. $V_{k+1}^*(s)$ represents the current V-function's optimal estimation, corresponding to the original V-value, which is estimated by V-function trained by Equation 3. We can iterate through the concept of V and Equation 8 to prove the conclusion of Equation 9(See Appendix G). For a static β, the final V-value is $V(s) = V^*(s) - \frac{1-\gamma^n}{1-\gamma}\beta$, in which n means that it takes n loops of the V function to converge.

Let's plug $V(s) = V^*(s) - \frac{1-\gamma^n}{1-\gamma}\beta$ into Equation 7 to consider the optimal Q-value:

$$Q(s,a) = r + \gamma V(s') - \beta = Q^*(s,a) - \frac{1-\gamma^{(n+1)}}{1-\gamma}\beta \tag{10}$$

V^* represents the final V-value estimated by the original V-function, and Q^* has a similar meaning. Therefore, we get our algorithm's theoretical Q-value and V-value, and our work shows that different environments have different n.

According to the above results, we get a relatively accurate Q-value and V-value of in-dataset state-action and a conservative Q-value of OOD actions. By adjusting the parameter β, we can change the degree of OOD actions' estimation and may have a higher probability of choosing OOD action.

Performance Analysis. We will provide an analysis of whether our algorithm can achieve state-of-the-art performance. First, for critics with V-function and Q-function, we know that it can recover the optimal value function under the dataset support constraints, which had been analysed in IQL. Therefore, our algorithm can sample the best action in the dataset.

However, we want our algorithm to explore appropriately to find a better policy distribution than an in-distribution optimal policy if it exists. We assume that better policy action is near the optimal policy action within the dataset and can be sampled if it exists. Our algorithm tries to reduce the Q-value of the action sampled by the current policy.

For optimal in-distribution action, our algorithm estimates the Q-value of it as $Q(s,a) = Q^*(s,a) - \frac{1-\gamma^{(n+1)}}{1-\gamma}\beta$. We assume that there is an OOD action which has a higher Q-value than the one above, i.e. $Q^*(s,a') > Q^*(s,a)$. For this action, the theoretical Q-value of our algorithm is $Q(s,a') =$

$Q^*(s,a') - \frac{1-\gamma^{(n_1+1)}}{1-\gamma})\beta$ (the rate of convergence of different Q may be different). As a result, we show that $Q(s,a')$ should be selected if $Q^*(s,a') > Q^*(s,a) + (\frac{1-\gamma^{(n+1)}}{1-\gamma} - \frac{1-\gamma^{(n_1+1)}}{1-\gamma})\beta$. Therefore, our algorithm has some exploratory ability to determine whether there is a better policy near the in-dataset optimal policy distribution to perform better.

In practice, as we show in sections 5.1 and 5.3, the actual Q-value floats around the theoretical Q-value. We analyse the difference bound between the actual Q-values and theoretical Q-values. At first, if most of the in-dataset actions are not fully sampled, the Q-value will be $Q^*(s,a) > \hat{Q}(s,a) > Q^*(s,a) - \frac{1-\gamma^{(n+1)}}{1-\gamma}\beta$. In addition, we analyse the final bound in Appendix C.

4.4 Implementation

Algorithm 1 flexible conservative estimate

```
Initialize V network $\psi$, Q network $\theta$, target Q network $\hat{\theta}$ and update target Q network $\hat{\theta} \leftarrow \theta$, action network $\phi$, set expectile regression parameters $\tau$, $\alpha$, and conservative parameter $\beta$ according to the dataset.
TD learning (DCE):
for each epoch do
    use Equation 4 to update the parameter of V-function $\psi$
    use Equation 6 to update the parameter of Q-function $\theta$ and update target network parameter $\hat{\theta} \leftarrow (1-\gamma)\hat{\theta} + \gamma\theta$
    If necessary, update $\alpha$ by SAC $\alpha$ loss and update $\beta$ by your update rule(in our algorithm, we use linear update every 50 epochs)
end for
Policy extraction:
for each epoch do
    Update policy by SAC policy loss
end for
```

We will discuss the implementation details of this algorithm. While our algorithm uses a similar critic loss objective, we do not use the advantage-weighted regression to train the actor. Our algorithm only makes a few simple changes to the general Actor-Critic. We use the standard SAC (Haarnoja et al. (2018)) policy for our work. I.e.the policy loss of our algorithm is as follows:

$$L_\pi(s) = E_{s_t \sim D}[E_{a'_t \sim \pi(\cdot|s_t;\phi)}\left[Q_\theta\left(s_t, a'_t\right) - \alpha \ln \pi\left(a'_t \mid s_t; \phi\right)\right]] \tag{11}$$

where ϕ is the policy parameter.

In addition, based on the IQL critique, our work introduces an additional loss to achieve flexible conservative estimation. Specifically, first, we train V-function and Q-function by using Equations 4 and 6. At the same time, we update the parameter α if the configure required (It is usually decided by the dataset and the relationship between α and β discussed in Appendix B). Then, we use Equation 11 to train the policy with the parameter α. Although we use static parameter β in the above subsection, we use a variable β to get better performance. Specifically, we change β when Q-function and V-function converges, and by selecting appropriate β for different dataset and environment, our algorithm can have state-of-the-art performance.

5 Experiments

In experiments, we demonstrate the performance of DCE in various settings and compare it to prior offline RL methods. First, we start with simple ablation experiments to show that our method has an effect and that the coefficient β can control the degree of conservatism in section 5.1. In addition, we also prove the correctness of the changes in the Q-value of in-dataset state-action and V values analyzed in the previous section. Then, we will compare DCE with state-of-the-art offline RL methods on the D4RL (Fu et al. (2020)) benchmark tasks in section 5.2.

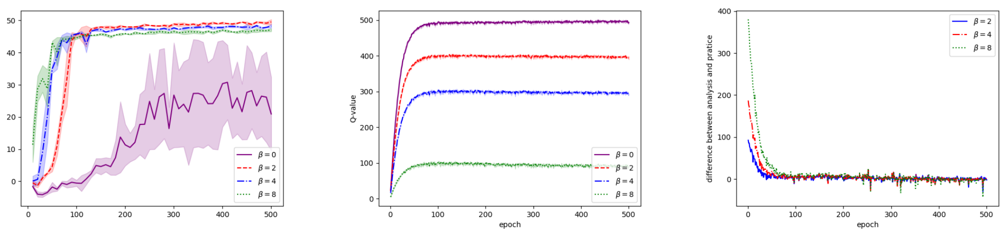

Figure 2: As an ablation experiment, our algorithm runs 500 epochs on a given random factor. **left:** We show the average normalized return of our algorithm with different parameters β in this sub-graph. **middle:** We show the average Q-value of in-dataset state-action for different β in this sub-graph. It is obvious that Q-value varies with the parameter β. **right:** We computer the difference between Q-value of in-dataset state-action and $r + \gamma V^* - \frac{1-\gamma^{(n+1)}}{1-\gamma}\beta$ and show the results on this sub-graph. This result shows that the Q-value of in-dataset state-action obeys our analysis in the previous section.

Before experiments, we try the different parameter τ to show if the influence of τ is different in our algorithm (see Figure 1). Figure 1 shows the performance of different τ in halfcheetah-medium, relocate-cloned, kitchen-mixed. The different values of τ affect the results of our algorithm. In addition, we have different degrees of influence of V-function for different environments, and we try to analyze this phenomenon in Appendix E. The value of τ had been discussed in IQL, so we set the same configuration of τ with IQL.

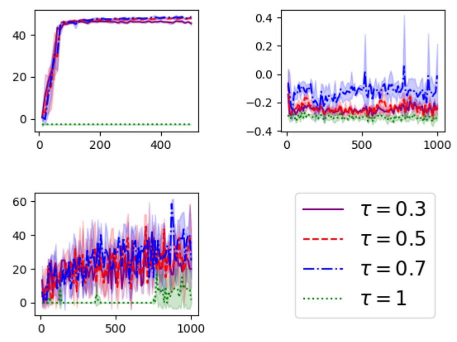

Figure 1: This figure shows the performance of different τ. **top left:** The performance in halfcheetah-medium. We can see that expect some values, different τ show little difference (Appendix E try analysis this phenomenon). **top right:** The performance in relocate-cloned. **lower left:** The performance in kitchen-mixed. By this figure, we find that in the Adroit and Kitchen task, the value of τ will impact the performance of our algorithm.

5.1 Double Conservative Estimation Of In-dataset Action

To verify that our penalty term is valid, we control for additional parameters to experiment. In this experiment, we use different β and the same seed as a compare condition. We fix this parameter for α in SAC and get $\alpha = 1.0$. To show the effect of our method, we set $\beta = 0, 2, 4, 8$ and demonstrate the performance of different β. For other parameters like t in V learning, we use the same sets with IQL. We run this network for 500000 gradients in halfcheetah-medium-v2 datasets, and the result is drawn as shown in Fig.2.

We can see that $\beta = 0$, this method has a poor performance even is lower than BC, and an inappropriate value of β, for example, 8, will have a poor performance too. However, even with the parameter $\beta = 0$, the policy performance is better than traditional SAC, which means the introduction of the V-function has a certain effect on preventing policy extrapolation. In addition, it is evident that the Q values of in-dataset state-action change proportionally follow β (seeing the middle of Fig.2). This means, as shown in the right-hand sub-graph in Fig.2, that the Q-value of in-distribution state-action varies with the value of parameter β and that the magnitude of the change conforms to our analysis above (i.e. the Q-value of in-dataset state-action fluctuates around $r + \gamma V(s) - \frac{1-\gamma^{(n+1)}}{1-\gamma}\beta$). And from that, we can deduce that the relationship between the value of V and V^* also satisfies the above analysis because

Dataset	BC	UWAC	PBRL	IQL	TD3-BC	CQL	**Ours**
HalfCheetah-r	2.1	2.3 ±0.0	11.0 ±5.8	13.7±5.3	11.0 ±5.8	17.5 ±1.5	**23.5±3.3**
Hopper-r	9.8	2.7±0.3	**26.8±9.3**	7.1±1.7	8.5±0.6	7.9±0.4	9.6±3
Walker2d-r	1.6	2.0±0.4	8.1±4.4	7.6±0.9	1.6±1.7	5.1±1.3	**11.3±6.6**
HalfCheetah-m	42.6	42.2 ±0.4	**57.9±1.5**	47.4±0.2	48.3 ±0.3	44.0±5.4	**57.4±1.3**
Hopper-m	52.9	50.9±4.4	**75.3±31.2**	66.2±5.7	59.3±4.2	58.5±2.1	**75.9±12.3**
Walker2d-m	75.3	75.4±3.0	**89.6±0.7**	78.3±8.7	83.7±2.1	72.5±0.8	84.7±4.5
HalfCheetah-m-r	36.6	35.9±3.7	45.1±8.0	44.2±1.2	44.6±0.5	45.5±0.5	**53.4±1.3**
Hopper-m-r	18.1	25.3±1.7	**100.6±1.0**	94.7±8.6	60.9±18.8	95.0±6.4	**99.5±8.6**
Walker2d-m-r	26.0	23.6±6.9	77.7±14.5	73.8±7.1	**81.8±5.5**	77.2±5.5	**82.3±11.7**
HalfCheetah-m-e	55.2	42.7±0.3	**92.3±1.1**	86.7±5.3	90.7±4.3	**91.6±2.8**	**92.7±1.6**
Hopper-m-e	52.5	44.9±8.1	**110.8±0.8**	91.5±14.3	98.0±9.4	105.4±6.8	91.3±18.4
Walker2d-m-e	107.5	96.5±9.1	**110.1±0.3**	**109.6±1.0**	**110.1±0.5**	108.8±0.7	**110.2±0.7**
MuJoCo total	480	444.4±38.3	**805.3±78.6**	720.8±60	698.5±53.7	729±34.2	**791.7±73.3**
Adroit total	93.9	34.8±11	116.1±9.418.7	118.1±30.7	0.0	93.6	**159.3±85.1**
Kitchen total	**154.5**	-	-	**159.8±22.6**	-	144.6	**156.8±32.8**
runtime(s)	6	176	-	10	-	18	12

Table 1: Average normalized score and the standard deviation of all algorithms over five seeds in the Gym. For every seed, we sample four trajectories and calculate the average return of these trajectories during the evaluation period. The highest-performing scores are highlighted. The score of UWAC, TD3-BC, PBRL, CQL, BC (except for the random environment), and IQL (except for the random environment) is the reported scores in Table 1 of PBRL (Bai et al. (2022)) and Table 1 of IQL (Kostrikov et al. (2021b)). The scores for other baselines are obtained by re-training with the 'v2' dataset of D4RL. Runtime shows each algorithm's computation time running 1000 iterations on RTX3070.

$V(s) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q(s,a))$. But, of course, the performance of policy change is negligible when the β has a large value.

5.2 Offline RL Benchmarks

In this subsection, we compare our algorithm with other offline RL algorithms in the D4RL benchmark (see Table 1). In D4RL, there are four types of MuJoCo tasks: Gym locomotion tasks, the Ant Maze tasks, the Adroit tasks and Kitchen robotic manipulation tasks. Our work compares the baseline algorithms on the Gym and Adroit environments, which have been used extensively in previous research. We compare our DCE with a series of representative algorithms: BC, UWAC (Wu et al. (2021)), PBRL (Bai et al. (2022)), IQL (Kostrikov et al. (2021b)), TD3-BC (Fujimoto & Gu (2021)), CQL (Kumar et al. (2020)). We analyse the performance of the algorithm on Gym locomotion tasks in this section and the algorithm's performance on Adroit tasks in Appendix A.

Result in Gym locomotion tasks. The Gym locomotion tasks include three environments: HalfCheetah, Hopper and Walker2D, and five dataset types for every environment: random, medium, medium-replay, medium-expert and expert. Our experiments run our algorithm on the first four datasets with the 'v2' type. We train each algorithm for one million time steps and record the average policy performance by interacting with the online environment. Table 1 reports the average normalized score corresponding to the performance of the different algorithms in each task. This table shows that PBRL performs best in most tasks among all the baseline algorithms, and CQL, IQL, and TD3-BC perform competitively in medium-expert datasets. Furthermore, we found that random seeds influence our algorithm in some datasets.

Through careful observation, we found that compared with baseline algorithms, our approach performs best on each dataset of the HalfCheetah, and the effect of random seed is small. In addition, compared with baseline algorithms, our algorithm usually has a better performance in the datasets without any optimal solution, including random, medium, and medium replay. Therefore, we believe proper exploration benefits datasets with suboptimal solutions. In addition, in the medium-expert dataset of each task, our algorithm has no significant advantage over PBRL, IQL, TD3-BC and CQL.

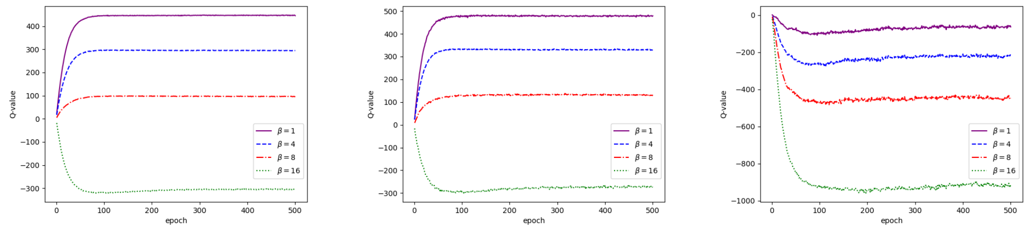

Figure 3: We train a traditional SAC (with V-function) algorithm without any additional penalty and add four critics with different values of β. We plotted the difference between the values of traditional SAC critics and the values of critics with different values of β. **left:** We draw the Q-values of action, which is a sample by additional policy for different parameters β. **middle:** The difference between max Q-values of the OOD action and Q-value of in-dataset action. **right:** We show the difference between the min Q-values of the OOD action and the Q-value of the in-distribution action.

5.3 Double Conservative Estimation Of OOD Action

We also compare the Q-value of OOD action for different parameters β to show that our work is also valid for OOD action and can adjust the degree of conservative estimation. We set $\beta = 1, 4, 8, 16$. In order to analyze the conservatism of Q-function to OOD state-action, we add two additional policies used to provide the OOD action. In addition, we use deterministic policy to ensure that the policy provides the same action for different parameters $beta$.

Specifically, we train our algorithm, while two additional policies do not require any training. One of their policies is the pre-trained best policy, and the other is the initialization policy. Their policies give a deterministic action for a state, and critics compute the corresponding Q-value. We compute the difference between the Q-value of OOD state-action generated by different β values and the Q-value of traditional SAC critic. The result is drawn as shown in Fig.3. For different parameters β, OOD state-action. We find that our algorithm can control the degree of conservative estimation by adjusting parameter β.

5.4 Runtime

Our algorithm has a relatively good computational speed while ensuring competitive performance. We measure runtime for our algorithm and CQL and IQL, which implementation in PyTorch both. By running those algorithms on NVIDIA GeForce RTX 3070, we found that our algorithm has faster computationally than CQL and is competitive computationally for IQL. Specifically, the CQL takes 18 seconds to perform 1000 gradient updates, and the IQL takes 10 seconds, while our algorithm needs 12 seconds (see Table 1).

6 Conclusion

In this paper, we propose a double conservative estimation, a conservative-based Model-free algorithm. This algorithm can change the degree of conservative estimation by controlling the parameter β to get better performance when the agent meets a different environment. In addition, this algorithm avoids the influence of OOD action on in-distribution action as much as possible while reducing the value of OOD state action. The introduction of a V-function conservative estimate of the Q-value of all state-action. Therefore we can see the V-function as a conservative estimation. We give a theoretical analysis of the influence of parameters on the Q-function and V-function and experiments that demonstrates the state-of-the-art performance by changing the degree of conservative estimation. In the experiment, parameter α of control policy exploration is influenced by the parameter β (Appendix B). However, our work lacks further analysis of the relationship between α and β. Therefore, our future research may focus on the relationship between the degree of conservative and policy exploration. In addition, trying to tune parameter β automatically is also a viable direction.

REFERENCES

Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. *Advances in neural information processing systems*, 21, 2008.

Chenjia Bai, Lingxiao Wang, Zhuoran Yang, Zhihong Deng, Animesh Garg, Peng Liu, and Zhaoran Wang. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *arXiv preprint arXiv:2202.11566*, 2022.

David Brandfonbrener, Will Whitney, Rajesh Ranganath, and Joan Bruna. Offline rl without off-policy evaluation. *Advances in Neural Information Processing Systems*, 34:4933–4946, 2021.

Rasool Fakoor, Jonas W Mueller, Kavosh Asadi, Pratik Chaudhari, and Alexander J Smola. Continuous doubly constrained batch reinforcement learning. *Advances in Neural Information Processing Systems*, 34:11260–11273, 2021.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. *international conference on machine learning*, 2020.

Caglar Gulcehre, Sergio Gómez Colmenarejo, Ziyu Wang, Jakub Sygnowski, Thomas Paine, Konrad Zolna, Yutian Chen, Matthew Hoffman, Razvan Pascanu, and Nando de Freitas. Regularized behavior value estimation. *arXiv preprint arXiv:2103.09575*, 2021.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33: 21810–21823, 2020.

Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pp. 5774–5783. PMLR, 2021a.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021b.

Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. *arXiv preprint arXiv:2112.04716*, 2021.

Byung-Jun Lee, Jongmin Lee, and Kee-Eung Kim. Representation balancing offline model-based reinforcement learning. In *International Conference on Learning Representations*, 2020.

Jongmin Lee, Wonseok Jeon, Byungjun Lee, Joelle Pineau, and Kee-Eung Kim. Optidice: Offline policy optimization via stationary distribution correction estimation. In *International Conference on Machine Learning*, pp. 6120–6130. PMLR, 2021.

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.

Xiaoteng Ma, Yiqin Yang, Hao Hu, Qihan Liu, Jun Yang, Chongjie Zhang, Qianchuan Zhao, and Bin Liang. Offline reinforcement learning with value-based episodic memory. *arXiv preprint arXiv:2110.09796*, 2021.

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv: Learning*, 2022.

Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *International conference on machine learning*, pp. 2701–2710. PMLR, 2017.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv: Learning*, 2021.

Aayam Shrestha, Stefan Lee, Prasad Tadepalli, and Alan Fern. Deepaveragers: offline reinforcement learning by solving derived non-parametric mdps. *arXiv preprint arXiv:2010.08891*, 2020.

Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

Núria Armengol Urpí, Sebastian Curi, and Andreas Krause. Risk-averse offline reinforcement learning. *arXiv preprint arXiv:2102.05371*, 2021.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33:7768–7778, 2020.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv: Learning*, 2019.

Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.

Dataset	BC	UWAC	PBRL	IQL	TD3-BC	CQL	**Ours**
Pen-human	34.4	10.1 ±3.2	35.4 ±3.3	71.5	0.0	37.5	**85.1**±**37**
Hammer-human	1.5	1.2±0.7	0.4±0.3	1.4	0.0	**4.4**	2.1±2.0
Door-human	0.5	0.4±0.2	0.1	4.3	0.0	**9.9**	4.8±14
Relocate-human	0.0	0.0	0.0	0.1	0.0	**0.2**	0.1±0.1
Pen-cloned	56.9	23.0 ±6.9	**74.9**±**9.8**	37.3	0.0	39.2	63.5±25.7
Hammer-cloned	0.8	0.4±0.0	0.8±0.5	**2.1**	0.0	**2.1**	1.0±0.6
Door-cloned	-0.1	0.0	**4.6**±**4.8**	1.6	0.0	0.4	2.9±5.6
Relocate-cloned	-0.1	-0.3	-0.1	-0.2	**0.0**	-0.1	-0.2±0.1
adroit total	93.9	34.8±11	116.1±18.7	118.1	0.0	93.6	**159.3**±**85.1**
kitchen-complete	**65.0**	-	-	**62.5**	-	43.8	50±10
kitchen-partial	38.0	-	-	46.3	-	49.8	**59** ±**14**
kitchen-mixed	**51.5**	-	-	**51.0**	-	**51.0**	48.8±8.8
kitchen total	**154.5**	-	-	**159.8**	-	144.6	**156.8**±**32.8**

Table 2: This table shows the average normalized score and the standard deviation of all algorithms over five seeds in the Gym. For every seed, we sample four trajectories and calculate the average return of these trajectories during the evaluation period. The highest-performing scores are highlighted. The score of UWAC, TD3-BC, PBRL, CQL, BC, and IQL is the reported scores in Table 4 of PBRL (Bai et al. (2022)) and Table 3 of IQL (Kostrikov et al. (2021b)). In addition, the total adroit average normalized score of IQL is 118.1 ± 30.7, and the total kitchen average normalized score of IQL is 159.8 ± 22.6 in their paper, but we did not find the exact value of each task.

A EXPERIMENTS IN ADROIT AND KITCHEN DOMAIN

The Adroit tasks include the four environments: Pen, Hammer, Door, and Relocate and three dataset types for every environment: human, clone and expert. Our experiments use two of those three datasets types: human and clone. The Kitchen tasks include complete, partial and mixed datasets. For every dataset, we use five different random seeds and sample five evaluation trajectories to average the mean performance of our algorithm. Our algorithm uses the standard offline reinforcement learning parameter and network structure. We use the same setting as IQL for parameter τ. Instead of IQL, our algorithm adds dropout to the Q-function in some datasets and then to policy in all datasets for Adroit tasks to get better performance.

Table 2 shows the performance of our algorithm and baseline offline reinforcement learning. Our algorithm has a better performance compared to the baseline algorithm. However, in some environments, our algorithm is sensitive to random seed (i.e. having a higher variance) because our algorithm has a constant value of parameter α. Setting a constant value of $alpha$ is when we use auto-update α (see Appendix B).

B RELATIONSHIP BETWEEN α AND β

Unlike the IQL algorithm, which uses the advantage-weighted regression as a policy, our algorithm uses the SAC algorithm's policy. Therefore, our algorithm needs to consider two parameters α and β, to get a better performance in a different environment and dataset. We add the α into a network to auto-update it in some datasets. In other datasets, we set the α as a constant value because of the relationship between α, β and the dataset.

In this section, we use auto-update α in the different datasets to show the influence of parameter β on the parameter α. We train our algorithm with different parameters β for 500 epochs. In addition, we train our algorithm with the same parameter β in different environments and datasets because it has some influence on the parameter α.

In fig.4, we demonstrate that the parameter α increases with the increase of parameter β within a specific range. In addition, for different environments and the same value of β, the value of α and the above range differ. In the right of fig.4, we can find that different environments will significantly influence the value of α.

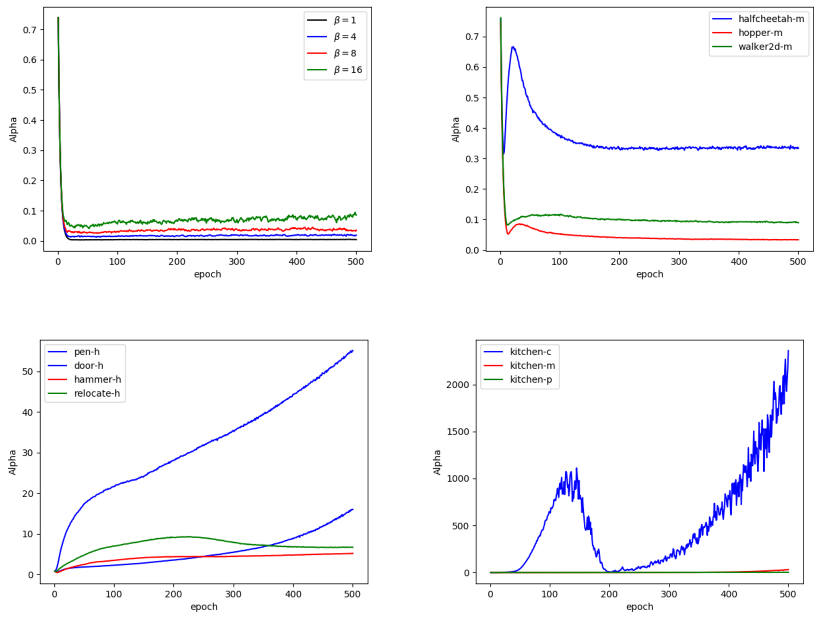

Figure 4: We show the value of auto-update α for different values of β and the dataset. **top left:**We draw the value of α for different β. **top right:**This sub-figure show the value of α in the MuJoCo environment. **lower left:**The value of α in the Adroit environment have some differences for the different task. **lower right:**In Kitchen environment,different datasets have vastly different α.

C BOUND OF ACTUAL Q-VALUE AND THEORETICAL Q-VALUE

We will analyse the difference between the Actual Q-value and Theoretical Q-value. Furthermore, before that, we show the sample-based version of Equation 6:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha \cdot \left(\sum_{\mathbf{s}\in\mathcal{D}} \mathbb{E}_{\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]\right) + \frac{1}{2|\mathcal{D}|} \sum_{\mathbf{s},\mathbf{a},\mathbf{s}'\in\mathcal{D}} \left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\mathbf{s},\mathbf{a})\right)^{2}\right] \tag{12}$$

where $\hat{\mathbb{B}^{\pi}}$ denotes the Bellman operator computed using dataset D:

$$\forall \mathbf{s},\mathbf{a} \in \mathcal{D}, \quad \left(\hat{\mathcal{B}}^{\pi}\hat{Q}^{k}\right)(\mathbf{s},\mathbf{a}) = r + \gamma \sum_{\mathbf{s}'} \hat{T}\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right) \hat{V}^{k}(s) \tag{13}$$

Furthermore, our algorithm uses expectile regression to learn V-value, which can give different weights for different Q-values with the parameter τ.

We consider the bound of difference between Q and Q^{*} in practice at first. By Equation 5 and Equation 6, we know that the difference between Q and Q^{*} is an additional penalty term. Therefore, the floating bound of they is influenced by the sampling error of the penalty term. The error of penalty term depends on the sampling frequency of policy and dataset, so we have:

$$\left|\hat{Q}(s,a) - \hat{Q}^{*}(s,a)\right| \leq \frac{n_{\mu}}{n_{\pi}} \times \frac{1-\gamma^{(n+1)}}{1-\gamma} \times \beta \tag{14}$$

Then we consider the low bound of the actual Q-value and theoretical Q-value. Following prior work Auer et al. (2008) Osband & Van Roy (2017) Kumar et al. (2020), We know two assumption as following:

$$|r - r(\mathbf{s},\mathbf{a})| \leq \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}}, \quad \left\|\hat{T}\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right) - T\left(\mathbf{s}' \mid \mathbf{s},\mathbf{a}\right)\right\|_{1} \leq \frac{C_{T,\delta}}{\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}} \tag{15}$$

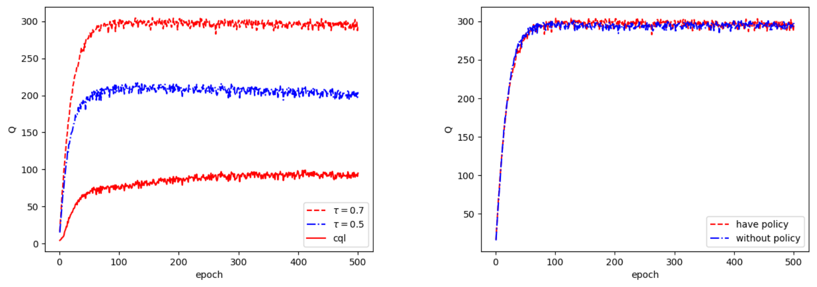

Figure 5: There are two sub-figure to show the experiment of Appendix D. **left:**We draw the Q-value, which is obtained by different configurations of τ and Equation 18 and show the difference with CQL. **right:** We show the influence of ODD action. Specifically, we train critics without policy, and the penalty term is the Q-value of the current state action.

Because our algorithm use expectile regression to learn V-value, we know the bound of V-value is $V(s) \leq \frac{2\tau R_{max}}{1-\gamma}$. By Equation 13 and 15, the difference between $\mathcal{B}^{\pi}\hat{Q}^{k}$ and $\hat{\mathcal{B}}^{\pi}\hat{Q}^{k}$ can be bounded:

$$\begin{aligned} \left|\left(\hat{\mathcal{B}}^{\pi}\hat{Q}^{k}\right)-\left(\mathcal{B}^{\pi}\hat{Q}^{k}\right)\right| &= \left|(r-r(\mathbf{s},\mathbf{a}))+\gamma\sum_{\mathbf{s}'}\left(\hat{T}\left(\mathbf{s}'\mid\mathbf{s},\mathbf{a}\right)-T\left(\mathbf{ss}'\mid\mathbf{s},\mathbf{a}\right)\right)V(s')\right| \\ &\leq |r-r(\mathbf{s},\mathbf{a})|+\gamma\left|\sum_{\mathbf{s}'}\left(\hat{T}\left(\mathbf{s}'\mid\mathbf{s},\mathbf{a}\right)-T\left(\mathbf{s}'\mid\mathbf{s},\mathbf{a}\right)\right)V(s')\right| \\ &\leq \frac{C_{r,\delta}+\gamma C_{T,\delta}2R_{\max}\tau/(1-\gamma)}{\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}} \end{aligned} \tag{16}$$

At last, we get the bound of the actual Q-value as follows:

$$\hat{Q}(s,a) \leq Q(s,a)+\frac{(1-\gamma^{(n+1)})n_{\mu}}{(1-\gamma)n_{\pi}}\beta+\frac{C_{r,\delta}+\gamma C_{T,\delta}2R_{\max}\tau/(1-\gamma)}{\sqrt{|\mathcal{D}(\mathbf{s},\mathbf{a})|}} \tag{17}$$

D Relationship with CQL

In CQL section 3.1, they propose Equation:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \alpha\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})]+\frac{1}{2}\mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a})-\hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\mathbf{s},\mathbf{a})\right)^{2}\right] \tag{18}$$

and analysis its lower bound. Because our algorithm is slightly different from this Equation, we need to analyse the specific difference and show the experiment. In Appendix C, we get the bound of our algorithm and compare it with CQL can know the difference between CQL and our algorithm.

We run Equation 18 by modifying the CQL, which sets the α of SAC as one and α of CQL as four. In addition, we use the same method as our algorithm for the penalty term rather than CQL. We compare this algorithm's Q-value with our algorithm's Q-value in Halfcheetah-medium-v2 (see Figure 5). We use our algorithm's different τ configured to show that parameter τ impacts the Q-value. As discussed in section 4.1, when the $\tau = 0.5$, expectile regression equals standard mean squared error loss. We find that a higher τ will lead to a slack conservative estimation. In addition, we run our algorithm without a policy in which current Q-values provide the penalty term. We avoid the sampling error of the penalty term in this way. We can find that Q-value without policy will be flatter because it reduces the sampling error.

Expect the Q-value of different Equations, and we draw the performance of their policy in different environments (see Figure 6). As Figure shows, in MuJoCo tasks, Equation 18 of CQL is better than our algorithm, but our algorithm performs better in Adroit and Kitchen tasks. We analyse this phenomenon in Appendix E.

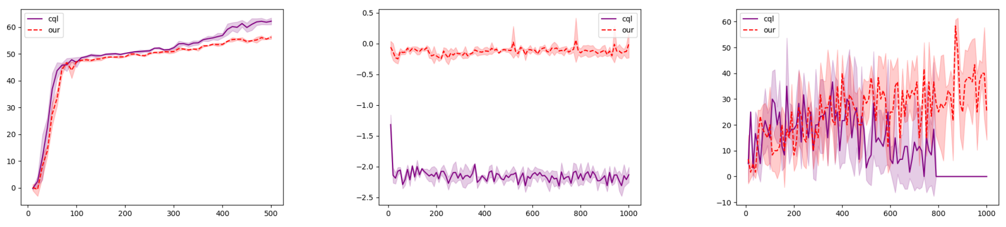

Figure 6: **left:** The performance in halfcheetah-medium. **meidum:** The performance in relocate-cloned. **right:** The performance in kitchen-mixed.

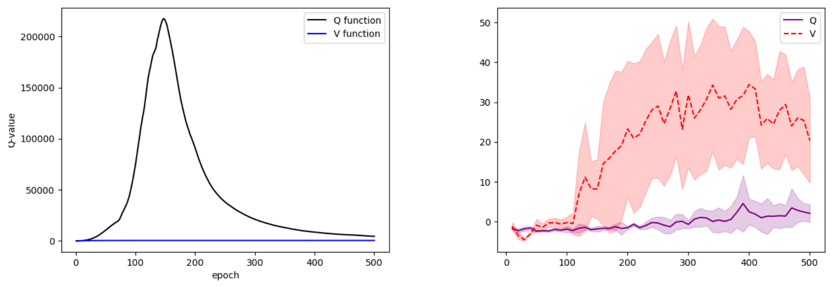

Figure 7: **left:** Q-value of in-distribution action in halfcheetah-medium. The Q-value of SAC without the V-function is 4000, and the Q-value of SAC with the V-function is 400. **right:** The performance for different SAC.

E V-FUNCTION IN CRITIC

In section 5, we find that parameter τ has varying degrees of influence on the performance of our algorithm. In Appendix D, the experiment shows that the agent without V-function performs better in MuJoCo tasks but worse in other tasks. We think it is caused by the V-function and the features of MuJoCo tasks.

First, V-function gives a lower value of the next state, leading to the conservative estimation of all actions. This conservative estimation limits policy exploration to a certain extent. To show this result, we train two SACs in halfcheetah-medium, one of which has a V-function (see Figure 7).

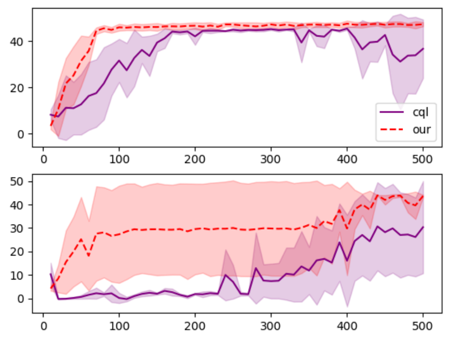

Figure 8: **left:** Average normalized return of $\alpha = 3$ **right:** Average normalized return of $\alpha = 5$.

Then, we think MuJoCo tasks have a wide ‘safe range’ in which OOD action will have less impact than other tasks. Therefore, a looser conservative estimate would perform better. We train our algorithm and Equation 18 of CQL with $\alpha = 3$, and we find that although performance declined, the performance of our algorithm is better). In addition, when $\alpha = 5$, Equation 18 of CQL will be invalid (see Figure 8). This is the exact opposite of a small α.

value	alpha	beta	update rate
HalfCheetah-r	auto	4-0.1	0.5
Hopper-r	auto	1-0.68	0.016
Walker2d-r	auto	0.1-0.08	0.009
HalfCheetah-m	auto	4-0.1	0.5
Hopper-m	auto	4-0.1	0.5
Walker2d-m	auto	4-3	0.05
HalfCheetah-m-r	auto	4-0.1	0.5
Hopper-m-r	auto	4-0.1	3.9
Walker2d-m-r	auto	4-1	0.15
HalfCheetah-m-e	auto	4-18	-14
Hopper-m-e	auto	6	0
Walker2d-m-e	auto	4-3	0.05
Pen-human	20	30	0
Hammer-human	auto	5	0
Door-human	0.5	4-0.5	0.25
Relocate-human	1	3	0
Pen-cloned	20	10	0
Hammer-cloned	4	4	0
Door-cloned	1	2	0
Relocate-cloned	1	0.1	0.5
kitchen-complete	0.5	2-1.7	0.015
kitchen-partial	0.5	2-1.65	0.0175
kitchen-mixed	0.5	1.5-1.15	0.0175

Table 3: Configure of our algorithm. Specially, we update the β every 50 epoch

Hyper-parameter	Value	Description.
Q-network	FC(256,256)	Full Connected layers with ReLU activations.
V-network	FC(256,256)	Full Connected layers with ReLU activations.
Actor	FC(256,256)	Full Connected layers with ReLU activations and Gaussian distribution.
lr of O-network	3e-4	Q-network learning rate.
lr of V-network	3e-4	V-network learning rate.
lr of actor	3e-4	Policy learning rate.
τ	0.7	The value of expectile regression.
Optimizer	Adam	Optimizer.
γ	0.99	Discount factor.
υ	0.005	Target network smoothing coefficient.

Table 4: The configuration of other parameters

F Configure Of Our algorithm

We give the parameters of our algorithm under different datasets in Table 3. The main parameters we modify are α and β, so for other parameters, we use the default value (see Table 4). In addition, we use the dropout for the critic in 'human' of Adroit tasks (value is the same as IQL). In practice, we set $\beta = 4$ to get a sub-optimal performance and adjust the value of β by this result.

G Proof of Equation 8 and 9

We set $\bigtriangledown L = 0$ for Equation 7, then we have:

$$\nabla_{\theta} Q(s,a) \times ((r(s,a) + \gamma V_{\psi}(s') - Q_{\theta}(s,a) + \beta) = 0 \tag{19}$$

So we get Equation 8. In addition, for Equation 9, we iterative prove it.

$$\begin{aligned}
V_0(s) &= V_0^*(s) \\
Q_0(s,a) &= r + \gamma V(s') - \beta \\
V_1(s) &= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_0(s,a)) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_0(s') - \beta) \\
&= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_0^*(s,a) - \beta) = V_0^*(s) - \beta \\
Q_1(s,a) &= r + \gamma V_1(s') - \beta = r + \gamma V_1^*(s\prime) - (1+\gamma)\beta \\
&\vdots \\
V_{k+1}(s) &= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_k(s,a)) = \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_k(s') - \beta) \\
&= \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_k^*(s') - (1 + \gamma + \gamma^2 + ... + \gamma^k)\beta) \\
&= \mathbb{E}_{(s,a)\sim\mathcal{D}}(r + \gamma V_k^*(s') - \frac{1-\gamma^k}{1-\gamma}\beta) \\
&= \mathbb{E}_{(s,a)\sim\mathcal{D}}(Q_k^*(s,a) - \frac{1-\gamma^k}{1-\gamma}\beta) = V_{k+1}^*(s) - \frac{1-\gamma^k}{1-\gamma}\beta
\end{aligned} \tag{20}$$