Section 9: Offline RL

Author: Seohong Park

1 What is Offline RL?

The goal of offline RL is to learn a policy $\pi(a \mid s)$ that maximizes the return *without* any interactions with the environment. Instead, we assume that we have access to a static dataset of transitions $\mathcal{D} = \{(s, a, r, s')\}$ previously collected by some policies (which may be suboptimal). This is an important problem setting in RL, because it is often costly or unsafe to collect online interactions in many real-world environments, such as healthcare, autonomous driving, and robotics.

Perhaps the first thought that comes to you might be: **why can't we just apply standard off-policy RL (*e.g.*, Q-learning) to the offline dataset?** For example, why can't we just run SAC on the replay buffer filled with the offline dataset $\mathcal{D}$? The answer is: you can, but it will likely explode.

Here's why. In Q-learning, we have the following Bellman backup:

$$Q_\theta(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s' \sim p(s' \mid s, a)} \Bigg[\underbrace{\max_{a' \in \mathcal{A}} Q_{\bar{\theta}}(s', a')}_{\textbf{can go OOD!}} \Bigg]. \tag{1}$$

The issue is that the a' that maximizes the learned target Q value may be out-of-distribution (OOD) of the dataset $\mathcal{D}$, which may overestimate the target value. This is not a very significant issue in *online* RL, because we can just execute this action in the environment and observe the consequences, which will eventually correct the Q values if they were overestimated. However, in offline RL, there is no such corrective feedback from the environment. So these overestimated Q values won't be corrected, and even worse, they will be propagated through the Bellman backup, which may cause the Q values to diverge eventually.

This distributional shift issue is the central challenge in offline RL. To address this, *hundreds* of papers have proposed a variety of algorithms. However, almost all offline RL methods share the **same** underlying principle. The slogan is:

"Maximize return while staying close to the dataset."

In other words, we want to *regularize* the learned policy to not go beyond what we know from the dataset while still maximizing the return. As a result, most offline RL algorithms have a hyperparameter that controls the strength of this regularization, which is often one of the most important hyperparameters to tune in offline RL.

2 Policy Constraint Methods

A straightforward way to implement the above principle is to add a behavioral regularization or constraint term to an existing off-policy RL algorithm, such as DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), or SAC (Haarnoja et al., 2018). Typically, these standard off-policy RL algorithms have the following actor loss for the policy $\pi_\theta(a \mid s)$:

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\theta(a \mid s)}[Q(s, a)], \tag{2}$$

where $Q(s, a)$ is a learned Q function.

To enforce the agent to stay close to the dataset, we can add a regularization term to the actor loss,

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s \sim \mathcal{D}} \Big[\mathbb{E}_{a \sim \pi_\theta(a \mid s)}[Q(s, a)] - \alpha D(\pi_\theta(\cdot \mid s), \pi_\beta(\cdot \mid s)) \Big], \tag{3}$$

or a constraint term to the optimization problem,

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\theta(a \mid s)}[Q(s, a)] \quad \text{subject to } D(\pi_\theta(\cdot \mid s), \pi_\beta(\cdot \mid s)) \leq \epsilon, \forall s. \tag{4}$$

Here π_β is the behavioral policy of the dataset $\mathcal{D}$, $D(\cdot, \cdot)$ is *some* divergence measure between policies, and α and ε are hyperparameters that control the strength of the regularization or constraint. These are general forms of the actor loss for *policy constraint* methods in offline RL.

A common choice of the divergence measure is the KL divergence, defined as follows:

$$D_{\mathrm{KL}}(p(x) \| q(x)) = \mathbb{E}_{x \sim p(x)} \left[\log \frac{p(x)}{q(x)} \right]. \tag{5}$$

Importantly, the KL divergence is asymmetric. So there are two possible ways to use it for behavioral regularization. The first is the *forward KL*,

$$D_{\mathrm{KL}}(\pi_\beta(\cdot \mid s) \| \pi_\theta(\cdot \mid s)) = \mathbb{E}_{a \sim \pi_\beta(a|s)} \left[\log \frac{\pi_\beta(a \mid s)}{\pi_\theta(a \mid s)} \right], \tag{6}$$

and the second is the *reverse KL*,

$$D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \| \pi_\beta(\cdot \mid s)) = \mathbb{E}_{a \sim \pi_\theta(a|s)} \left[\log \frac{\pi_\theta(a \mid s)}{\pi_\beta(a \mid s)} \right]. \tag{7}$$

While they both are based on KL divergences, they have quite different properties. The forward KL is *mode covering*, which encourages the policy to cover all modes of the behavioral distribution. The reverse KL is *mode seeking*, which encourages the learned policy to approximate one of the modes of the behavioral distribution.

You don't need to memorize which one is which; you just need to look at the objective. For example, in the forward KL (Equation (6)), when $\pi_\theta(a \mid s) = 0$, we must have $\pi_\beta(a \mid s) = 0$ because otherwise the KL divergence will be infinite. This means that the learned policy must not miss any mode of the behavioral distribution. So it is mode covering. You can do the same reasoning for the reverse KL to see that it is mode seeking.

Between the two, we ideally want to use the reverse KL (mode-seeking) for offline RL, because we don't want to cover all modes of the behavioral distribution, some of which may be suboptimal. However, the forward KL is often easier to implement in practice (as we will see below) and often works reasonably well, so it is also widely used in practice.

2.1 AC+BC

If we use the forward KL for behavioral regularization, we get an "actor-critic + behavioral cloning" (AC+BC) algorithm, which is arguably one of the simplest ways to implement offline RL. This includes TD3+BC (Fujimoto and Gu, 2021), DDPG+BC, and SAC+BC. The actor loss of AC+BC is as follows:

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s \sim \mathcal{D}} \Big[\mathbb{E}_{a \sim \pi_\theta(a|s)}[Q(s,a)] - \alpha D_{\mathrm{KL}}(\pi_\beta(\cdot \mid s), \pi_\theta(\cdot \mid s)) \Big] \tag{8}$$

$$= \arg\max_\theta \mathbb{E}_{s \sim \mathcal{D}} \Big[\mathbb{E}_{a \sim \pi_\theta(a|s)}[Q(s,a)] + \alpha \mathbb{E}_{a \sim \pi_\beta(a|s)}[\log \pi_\theta(a \mid s)] \Big] + (\text{const}), \tag{9}$$

where we ignored the constant term $\alpha \mathbb{E}_{a \sim \pi_\beta(a|s)}[\log \pi_\beta(a \mid s)]$, which independent of θ.

Intuitively, this is nothing other than adding a standard BC loss to the actor loss of an existing off-policy RL algorithm! Everything else, such as the critic loss, remains exactly the same as the original off-policy RL algorithm. Despite its simplicity, this approach is often surprisingly strong in practice and is widely used in practice. One downside of this approach is that it requires tuning the BC coefficient α, which is generally quite sensitive and may differ substantially across tasks. You will see (or may already have seen) this in Homework 5.

2.2 AWR

If we use the reverse KL for the behavioral constraint, we get the advantage-weighted regression (AWR) algorithm (Peng et al., 2019). The constrained optimization problem in AWR is as follows:

$$\max_\theta \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\theta(a|s)}[Q(s,a)] \quad \text{subject to } D_{\mathrm{KL}}(\pi_\theta(\cdot \mid s) \| \pi_\beta(\cdot \mid s)) \leq \epsilon, \forall s. \tag{10}$$

This can be analytically solved via the method of Lagrange multipliers, which gives the following closed-form solution for the optimal (constrained) policy:

$$\pi^*(a \mid s) = \frac{1}{Z(s)} \exp\left(\frac{1}{\lambda} Q(s,a)\right) \pi_\beta(a \mid s) \tag{11}$$

$$\propto \exp\left(\frac{1}{\lambda} Q(s,a)\right) \pi_\beta(a \mid s), \tag{12}$$

where λ is the Lagrange multipler for the KL constraint, and $Z(s)$ is the partition function that normalizes the distribution. If you haven't derived this before, it's a **very** good exercise to work through it yourself by hand!

Given this closed-form solution for the optimal policy, we can train a parameterized policy $\pi_\theta(a \mid s)$ by minimizing the *forward* KL to the optimal policy $\pi^*(a \mid s)$,

$$\theta \leftarrow \arg\min_\theta \mathbb{E}_{s\sim\mathcal{D}}[D_{\mathrm{KL}}(\pi^*(\cdot \mid s) \| \pi_\theta(\cdot \mid s))] \tag{13}$$

$$= \arg\max_\theta \mathbb{E}_{s,a\sim\mathcal{D}} \left[\frac{1}{Z(s)} \exp\left(\frac{1}{\lambda} Q(s,a)\right) \log \pi_\theta(a \mid s)\right], \tag{14}$$

where the derivation details are (again) left as an exercise. Since $Q(s,a) = A(s,a) + V(s)$, this can be equivalently written in terms of advantages as follows:

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s,a\sim\mathcal{D}} \left[\frac{1}{Z'(s)} \exp\left(\frac{1}{\lambda} A(s,a)\right) \log \pi_\theta(a \mid s)\right], \tag{15}$$

where $Z'(s)$ is a different partition function. In practice, we often just ignore the partition function ($Z(s)$ or $Z'(s)$) because it only re-weights the loss for different states and doesn't affect the optimal action distribution at each state. This leads to the following simplified objective called advantage-weighted regression (AWR):

$$\theta \leftarrow \arg\max_\theta \mathbb{E}_{s,a\sim\mathcal{D}} \left[\exp\left(\frac{1}{\lambda} A(s,a)\right) \log \pi_\theta(a \mid s)\right]. \tag{16}$$

This loss has a very intuitive interpretation: it is simply weighted behavioral cloning, where the weights are given by the exponentiated advantages. In other words, it makes the policy selectively mimic only the good behaviors in the dataset based on the learned advantages. (Do you think we can replace the exponentiated advantages with the advantages themselves in practice? Why or why not?) Note that we used *both* the reverse and forward KL in the derivation of AWR.

In practice, AWR is often easier to tune than AC+BC, because it is essentially based on (weighted) supervised learning. However, it is sometimes not as performant as fully tuned AC+BC. There is a trade-off between performance and tunability between these two approaches.

3 Implicit Q-Learning

Implicit Q-learning (IQL) (Kostrikov et al., 2022) provides a different way to prevent the out-of-distribution action issue in offline RL. The key idea of IQL is to simply *never* query the Q function with out-of-distribution actions, while maximizing the return.

Naïvely, we can achieve this by modifying the original Bellman backup,

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} \left[\max_{a' \in \mathcal{A}} Q(s',a')\right], \tag{17}$$

into the following *in-sample* Bellman backup:

$$Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} \left[\max_{a':(s',a')\in\mathcal{D}} Q(s',a')\right]. \tag{18}$$

That is, we take the maximum only over the actions that are in the dataset $\mathcal{D}$, so that $Q(s', a')$ is never evaluated at out-of-distribution actions a'.

Unfortunately, this naïve approach will **not** work in practice. The problem is that, in continuous-control environments, we typically have *exactly one* action for each state in the dataset, because usually no two states are exactly the same in a continuous state space. So the maximum over actions in the Bellman backup is essentially meaningless in the naïve formulation above.

The main idea of IQL lies in how to approximate this in-sample maximization in practice. The first step is to separate the expectation $\mathbb{E}_{s' \sim p(s'|s,a)}$ and the maximum $\max_{a':(s',a') \in \mathcal{D}}$ in the Bellman backup by additionally introducing a value function $V(s)$, as follows:

$$Q(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \left[V(s') \right], \tag{19}$$

$$V(s) \leftarrow \max_{a:(s,a) \in \mathcal{D}} Q(s, a). \tag{20}$$

Now, the first equation is a standard Bellman backup for the Q function. This can be trained with standard regression:

$$Q \leftarrow \arg\min_{Q} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(Q(s, a) - (r(s, a) + \gamma V(s')) \right)^2 \right]. \tag{21}$$

The interesting part is the second equation. IQL approximates the maximum over actions with an *expectile regression*:

$$V \leftarrow \arg\min_{V} \mathbb{E}_{(s,a) \sim \mathcal{D}} \left[\ell_2^\tau (V(s) - Q(s, a)) \right], \tag{22}$$

where the expectile regression loss ℓ_2^τ is defined as

$$\ell_2^\tau(x) = \begin{cases} (1 - \tau)x^2 & \text{if } x > 0 \\ \tau x^2 & \text{else,} \end{cases} \tag{23}$$

and $\tau \in [0.5, 1)$ is the expectile hyperparameter that controls the degree of approximation to the maximum. Intuitively, this expectile regression is an asymmetric loss that penalizes underestimation more than overestimation (think about what happens when $V(s) < Q(s, a)$ and when $V(s) > Q(s, a)$). Note that $\tau = 0.5$ corresponds to the standard mean regression, and $\tau \to 1$ corresponds to the hard maximum. Since this loss *implicitly* approximates the maximum over actions via expectile regression, it does not suffer from the issue of having only one action per state in the dataset, unlike the naïve formulation above.

After training the Q function and the value function with the above losses (Equations (21) and (22)), we can extract a policy $\pi_\theta(a \mid s)$ from the learned Q function. The original IQL paper uses AWR for policy extraction, but we can use any other policy extraction method as well, such as AC+BC.

4 Conservative Q-Learning

Another principle in offline RL is *pessimism*, which is the idea of learning a *conservative* value function that underestimates the value of out-of-distribution actions. As a result, the agent will be discouraged from taking out-of-distribution actions, mitigating the distributional shift issue.

Conservative Q-learning (CQL) (Kumar et al., 2020) is a representative algorithm that implements this principle. CQL begins by adding a conservative regularization term to the original Bellman backup:

$$\begin{aligned} Q \leftarrow \arg\min_{Q} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(Q(s, a) - \left(r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(a'|s')} [Q(s', a')] \right) \right)^2 \right] \\ + \alpha \mathbb{E}_{s \sim \mathcal{D}} \left[\mathbb{E}_{a \sim \mu(a|s)} [Q(s, a)] \right], \end{aligned} \tag{24}$$

where $\mu(a \mid s)$ is some action distribution that covers the entire action space (e.g., uniform distribution) and α is a hyperparameter that controls the strength of the regularizer. Intuitively, this regularization term pushes down the Q values of all actions, especially those that are not in the dataset.

However, this will also push down the Q values of in-distribution (ID) actions, which is undesirable. To address this, CQL adds an additional term to push up the Q values of in-distribution actions:

$$Q \leftarrow \arg\min_Q \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left(Q(s,a) - \left(r(s,a) + \gamma \mathbb{E}_{a'\sim\pi(a'|s')}[Q(s',a')] \right) \right)^2 \right] + \alpha \underbrace{\mathbb{E}_{s\sim\mathcal{D}} \left[\mathbb{E}_{a\sim\mu(a|s)}[Q(s,a)] \right]}_{\text{push down OOD actions}} - \alpha \underbrace{\mathbb{E}_{(s,a)\sim\mathcal{D}}[Q(s,a)]}_{\text{push up ID actions}}. \tag{25}$$

This is the basic form of the CQL algorithm. There are a few practical variants of CQL that use different regularization terms, such as using the log-sum-exp of Q values instead of the uniform distribution for $\mu(a \mid s)$ for pushing down OOD actions. These variants are often more performant than the basic form of CQL.

While IQL and CQL look quite different from the policy constraint methods we discussed above, they (or their variants) can be shown to be equivalent to certain variants of policy constraint methods with particular choices of the divergence measure (Garg et al., 2023; Sikchi et al., 2024). For example, under certain conditions, CQL is mathematically equivalent to adding a χ^2-divergence regularizer between the learned policy and the behavioral policy (although they have different training objectives in practice).

References

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, 2018.

Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent rl without entropy. In *International Conference on Learning Representations (ICLR)*, 2023.

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 (ICML)*, 2018.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Aviral Kumar, Aurick Zhou, G. Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *ArXiv*, abs/1910.00177, 2019.

Harshit S. Sikchi, Qinqing Zheng, Amy Zhang, and Scott Niekum. Dual rl: Unification and new methods for reinforcement and imitation learning. In *International Conference on Learning Representations (ICLR)*, 2024.