

A PROOFS

A.1 PROOF OF LEMMA 2

Proof. We can rewrite $V_{\tau_2}(s)$ as

$$\begin{aligned}
 V_{\tau_1}(s) &= \mathbb{E}_{a \sim \mu(\cdot|s)}^{\tau_1} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} [V_{\tau_1}(s')]] \\
 &\leq \mathbb{E}_{a \sim \mu(\cdot|s)}^{\tau_2} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} [V_{\tau_1}(s')]] \\
 &= \mathbb{E}_{a \sim \mu(\cdot|s)}^{\tau_2} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} \mathbb{E}_{a' \sim \mu(\cdot|s')}^{\tau_1} [r(s', a') + \gamma \mathbb{E}_{s'' \sim p(\cdot|s', a')} [V_{\tau_1}(s'')]]] \\
 &\leq \mathbb{E}_{a \sim \mu(\cdot|s)}^{\tau_2} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} \mathbb{E}_{a' \sim \mu(\cdot|s')}^{\tau_2} [r(s', a') + \gamma \mathbb{E}_{s'' \sim p(\cdot|s', a')} [V_{\tau_1}(s'')]]] \\
 &= \mathbb{E}_{a \sim \mu(\cdot|s)}^{\tau_2} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s, a)} \mathbb{E}_{a' \sim \mu(\cdot|s')}^{\tau_2} [r(s', a') + \gamma \mathbb{E}_{s'' \sim p(\cdot|s', a')} \mathbb{E}_{a'' \sim \mu(\cdot|s'')}^{\tau_1} [r(s'', a'') + \dots]]] \\
 &\vdots \\
 &\leq V_{\tau_2}(s)
 \end{aligned}$$

□

B EXPERIMENTAL DETAILS

Comparisons and baselines. We compare to methods that are representative of both multi-step dynamic programming and one-step approaches. In the former category, we compare to CQL (Kumar et al., 2020), TD3+BC (Fujimoto & Gu, 2021), and AWAC (Nair et al., 2020). In the latter category, we compare to Onestep RL (Brandfonbrener et al., 2021) and Decision Transformers (Chen et al., 2021). We obtained the Decision Transformers results on Ant Maze subsets of D4RL tasks using the author-provided implementation² and following authors instructions communicated over email. We obtained results for TD3+BC and Onestep RL (Exp. Weight) directly from the authors. The results previously reported for CQL on D4RL mujoco tasks correspond to the “-v0” versions of the tasks. We obtained results for “-v2” datasets using an author-suggested implementation.³

Experimental details. For the MuJoCo locomotion tasks, we average mean returns over 10 evaluation trajectories and over 5 random seeds. For the Ant Maze tasks, we average over 100 evaluation trajectories. We use $\tau = 0.9$ and $\beta = 10.0$ for antmaze tasks and $\tau = 0.7$ and $\beta = 0.2$ for mujoco tasks. We use Adam optimizer with a learning rate $3 \cdot 10^{-4}$ and 2 layer MLP with ReLU activations and 256 hidden units for all networks. We parameterize the policy as a Gaussian distribution with a state-independent standard deviation. We update the target network with soft updates with parameter $\alpha = 0.005$. And following Brandfonbrener et al. (2021) we clip exponentiated advantages to $(-\infty, 100]$. We implemented our method in the JAX framework using the Flax neural networks library.

Runtime. Our approach is also notably computationally faster than the baselines. For example, our implementation takes 20 minutes on one GTX1080 GPU to perform 1M updates, while CQL, the closest baseline in terms of results, takes more than 80 minutes. Our reimplement of TD3+BC takes 20 minutes as well (Fujimoto et al., 2018); however, this method’s performance closely tracks that of other single-step methods which, as we show, are generally not capable of performing multi-step dynamic programming, as evidenced by their poor results on the Ant Maze tasks, and do not outperform prior multi-step methods (such as CQL) on the locomotion tasks.

²<https://github.com/kzl/decision-transformer>

³<https://github.com/young-geng/CQL>

⁴Note that Chen et al. (2021) and Brandfonbrener et al. (2021) incorrectly report results for some prior methods, such as CQL, using the “-v0” environments. These generally produce lower scores than the “-v2” environments that these papers use for their own methods. We use the “-v2” environments for all methods to ensure a fair comparison, resulting in higher values for some prior approaches. Because of this fix, our reported CQL scores are higher than all other prior methods.

C CONNECTIONS TO PRIOR WORK

In this section, we discuss how our approach is related to prior work on offline reinforcement learning. In particular, we discuss connections to BCQ (Fujimoto et al., 2019).

Our batch constrained optimization objective is similar to BCQ (Fujimoto et al., 2019). In particular, the authors of BCQ build on the Q-learning framework and define the policy as

$$\pi(s) = \arg \max_a Q(s, a). \quad (7)$$

$s.t. (s, a) \in \mathcal{D}$

Note that in contrast to the standard Q-learning, maximization in Eqn. (7) is performed only over the state-action pairs that appear in the dataset. In (Fujimoto et al., 2019), these constraints are implemented via fitting a generative model $\mu(\cdot|s)$ on the dataset, sampling several candidate actions from this generative model, and taking an argmax over these actions:

$$\pi(s) = \arg \max_{\{a_i | a_i \sim \mu(\cdot|s), i=1 \dots N\}} Q(s, a_i).$$

However, this generative model can still produce out-of-dataset actions that will lead to querying undefined Q-values. Thus, our work introduces an alternative way to optimize this objective without requiring an additional density model. Our approach avoids this issue by enforcing the hard constraints via estimating expectiles. Also, it is worth mentioning that a number of sampled actions N in BCQ has similar properties to choosing a particular expectile τ in our approach.

Note that our algorithm for optimal value approximation does not require an explicit policy, in contrast to other algorithms for offline reinforcement learning for continuous action spaces (Fujimoto et al., 2019; Fujimoto & Gu, 2021; Wu et al., 2019; Kostrikov et al., 2021; Kumar et al., 2019; 2020). Thus, we do not need to alternate between actor and critic updates, though with continuous actions, we must still extract an actor at the end once the critic converges.