A Minimalist Approach to Offline Reinforcement Learning

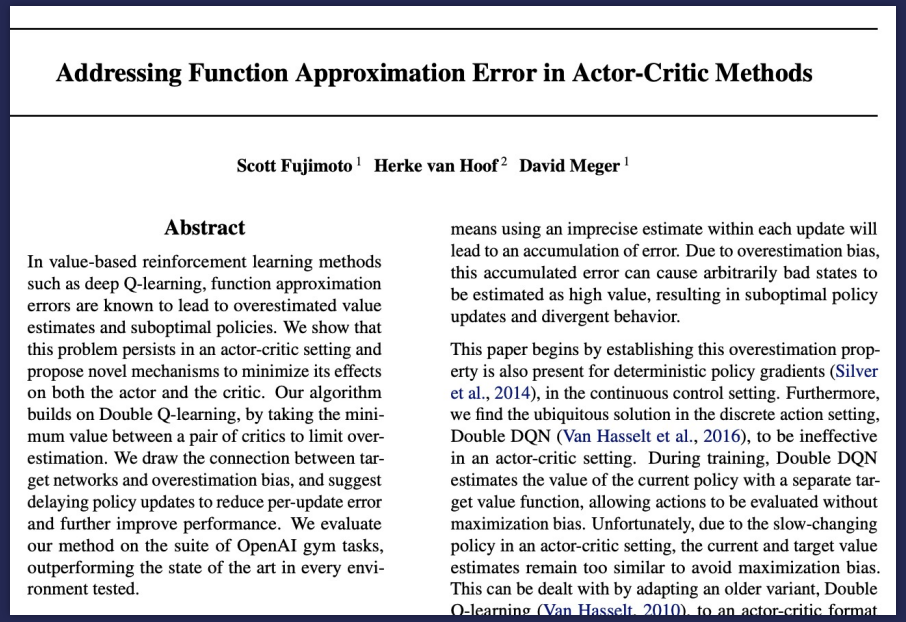
Fujimoto et al. *NeurIPS 2021 Spotlight*

Hyeonhoon Lee, MD(DKM), PhD
jackli0373@gmail.com

Contents

- **Introduction**
  - Offline RL
  - Challenges in offline RL algorithm
  - Criticism to current offline RL algorithm
- **A minimalist offline RL algorithm**
  - TD3 + BC (behavior cloning)
  - Normalized states
- **Experiments**

Papers



Addressing Function Approximation Error in Actor-Critic Methods

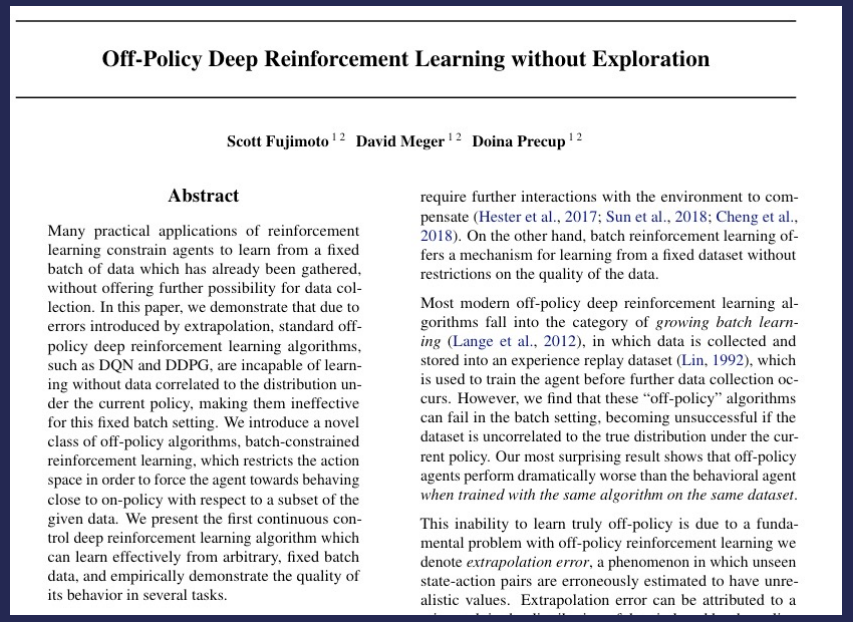
Scott Fujimoto [1] Herke van Hoof [2] David Meger [1]

Abstract

In value-based reinforcement learning methods such as deep Q-learning, function approximation errors are known to lead to overestimated value estimates and suboptimal policies. We show that this problem persists in an actor-critic setting and propose novel mechanisms to minimize its effects on both the actor and the critic. Our algorithm builds on Double Q-learning, by taking the minimum value between a pair of critics to limit overestimation. We draw the connection between target networks and overestimation bias, and suggest delaying policy updates to reduce per-update error and further improve performance. We evaluate our method on the suite of OpenAI gym tasks, outperforming the state of the art in every environment tested.

means using an imprecise estimate within each update will lead to an accumulation of error. Due to overestimation bias, this accumulated error can cause arbitrarily bad states to be estimated as high value, resulting in suboptimal policy updates and divergent behavior.

This paper begins by establishing this overestimation property is also present for deterministic policy gradients (Silver et al., 2014), in the continuous control setting. Furthermore, we find the ubiquitous solution in the discrete action setting, Double DQN (Van Hasselt et al., 2016), to be ineffective in an actor-critic setting. During training, Double DQN estimates the value of the current policy with a separate target value function, allowing actions to be evaluated without maximization bias. Unfortunately, due to the slow-changing policy in an actor-critic setting, the current and target value estimates remain too similar to avoid maximization bias. This can be dealt with by adapting an older variant, Double Q-learning (Van Hasselt, 2010), to an actor-critic format



Off-Policy Deep Reinforcement Learning without Exploration

Scott Fujimoto [1 2] David Meger [1 2] Doina Precup [1 2]

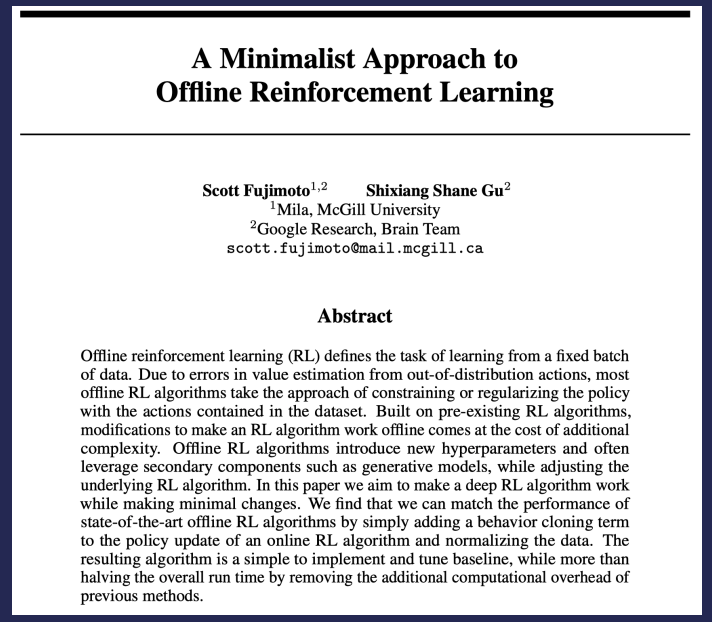
Abstract

Many practical applications of reinforcement learning constrain agents to learn from a fixed batch of data which has already been gathered, without offering further possibility for data collection. In this paper, we demonstrate that due to errors introduced by extrapolation, standard off-policy deep reinforcement learning algorithms, such as DQN and DDPG, are incapable of learning without data correlated to the distribution under the current policy, making them ineffective for this fixed batch setting. We introduce a novel class of off-policy algorithms, batch-constrained reinforcement learning, which restricts the action space in order to force the agent towards behaving close to on-policy with respect to a subset of the given data. We present the first continuous control deep reinforcement learning algorithm which can learn effectively from arbitrary, fixed batch data, and empirically demonstrate the quality of its behavior in several tasks.

require further interactions with the environment to compensate (Hester et al., 2017; Sun et al., 2018; Cheng et al., 2018). On the other hand, batch reinforcement learning offers a mechanism for learning from a fixed dataset without restrictions on the quality of the data.

Most modern off-policy deep reinforcement learning algorithms fall into the category of *growing batch learning* (Lange et al., 2012), in which data is collected and stored into an experience replay dataset (Lin, 1992), which is used to train the agent before further data collection occurs. However, we find that these "off-policy" algorithms can fail in the batch setting, becoming unsuccessful if the dataset is uncorrelated to the true distribution under the current policy. Our most surprising result shows that off-policy agents perform dramatically worse than the behavioral agent *when trained with the same algorithm on the same dataset*.

This inability to learn truly off-policy is due to a fundamental problem with off-policy reinforcement learning we denote *extrapolation error*, a phenomenon in which unseen state-action pairs are erroneously estimated to have unrealistic values. Extrapolation error can be attributed to a



A Minimalist Approach to Offline Reinforcement Learning

Scott Fujimoto[1,2] Shixiang Shane Gu[2]
[1]Mila, McGill University
[2]Google Research, Brain Team
scott.fujimoto@mail.mcgill.ca

Abstract

Offline reinforcement learning (RL) defines the task of learning from a fixed batch of data. Due to errors in value estimation from out-of-distribution actions, most offline RL algorithms take the approach of constraining or regularizing the policy with the actions contained in the dataset. Built on pre-existing RL algorithms, modifications to make an RL algorithm work offline comes at the cost of additional complexity. Offline RL algorithms introduce new hyperparameters and often leverage secondary components such as generative models, while adjusting the underlying RL algorithm. In this paper we aim to make a deep RL algorithm work while making minimal changes. We find that we can match the performance of state-of-the-art offline RL algorithms by simply adding a behavior cloning term to the policy update of an online RL algorithm and normalizing the data. The resulting algorithm is a simple to implement and tune baseline, while more than halving the overall run time by removing the additional computational overhead of previous methods.

TD3 → BCQ → Current

Offline RL

- A.k.a. batch RL
- The agent learns from a fixed-sized dataset, collected by some arbitrary and possibly unknown process.
- *No interaction with environment*

c.f. https://danieltakeshi.github.io/2020/06/28/offline-rl/ [ENG]

c.f. https://talkingaboutme.tistory.com/entry/RL-Offline-Reinforcement-Learning [KOR]

Challenges in offline RL

- *Extrapolation error : an error in policy evaluation*

Absent Data. If any state-action pair (s, a) is unavailable, then error is introduced as some function of the amount of similar data and approximation error. This means that the estimate of $Q_\theta(s', \pi(s'))$ may be arbitrarily bad without sufficient data near $(s', \pi(s'))$.

Model Bias. When performing off-policy Q-learning with a batch $\mathcal{B}$, the Bellman operator $\mathcal{T}^\pi$ is approximated by sampling transitions tuples (s, a, r, s') from $\mathcal{B}$ to estimate the expectation over s'. However, for a stochastic MDP, without infinite state-action visitation, this produces a biased estimate of the transition dynamics:

$$\mathcal{T}^\pi Q(s, a) \approx \mathbb{E}_{s' \sim \mathcal{B}}[r + \gamma Q(s', \pi(s'))], \tag{4}$$

where the expectation is with respect to transitions in the batch $\mathcal{B}$, rather than the true MDP.

Training Mismatch. Even with sufficient data, in deep Q-learning systems, transitions are sampled uniformly from the dataset, giving a loss weighted with respect to the likelihood of data in the batch:

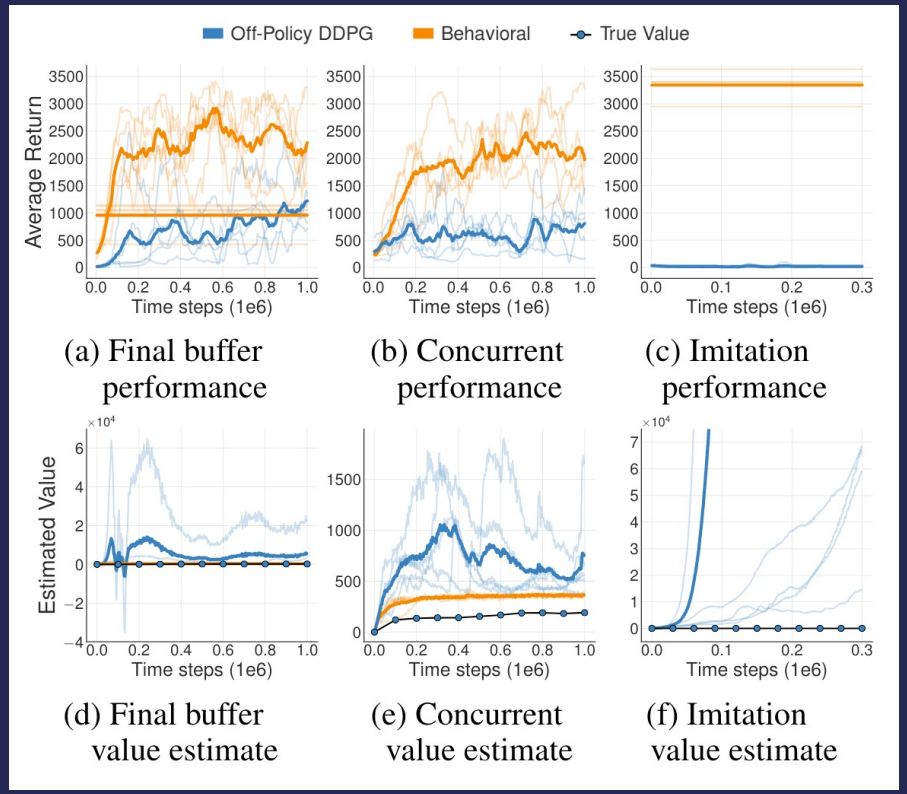
$$\approx \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s') \in \mathcal{B}} ||r + \gamma Q_{\theta'}(s', \pi(s')) - Q_\theta(s, a)||^2. \tag{5}$$

If the distribution of data in the batch does not correspond with the distribution under the current policy, the value function may be a poor estimate of actions selected by the current policy, due to the mismatch in training.

Fujimoto et al. 2019b (BCQ)

Challenges in offline RL

- Experiments with DDPG
  - Batch 1: Final buffer
  - Batch 2: Concurrent (buffer replay)
  - Batch 3: Imitation (A trained DDPG agent)
- *Extrapolation error can be highly detrimental to learning off-policy in a batch reinforcement learning setting.*



Fujimoto et al. 2019b (BCQ)

Criticism to current offline RL algorithms

- 1. Implementation and Tuning Complexities

	CQL [Kumar et al., 2020]	Fisher-BRC [Kostrikov et al., 2021]	TD3+BC (Ours)
Algorithmic Adjustments	Add regularizer to critic†	Train a generative model†‡	Add a BC term†
	Approximate logsumexp with sampling‡	Replace critic with offset function	
		Gradient penalty on offset function†	
Implementation Adjustments	Architecture†‡	Architecture†‡	Normalize states
	Actor learning rate†	Reward bonus†	
	Pre-training actor	Remove SAC entropy term	
	Remove SAC entropy term		
	Max over sampled actions‡		

Table 1: Implementation changes offline RL algorithms make to the underlying base RL algorithm. † corresponds to details that add additional hyperparameter(s), and ‡ corresponds to ones that add a computational cost.

- Both methods modify SAC by removing the entropy term in the target update and modify the default network architecture → *Add supplementary hyperparameters (+ computational costs)*

Criticism to current offline RL algorithms

- 2. Extra computation requirement

	CQL (GitHub)	Fisher-BRC (GitHub)	Fisher-BRC (Ours)	TD3+BC (Ours)
Implementation	25m	39m	15m	< 1s
Algorithmic	1h 29m	33m	58m	< 5s
Total	4h 11m	2h 12m	2h 8m	39m

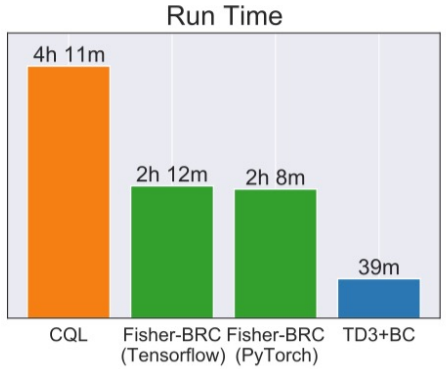


Table 3: Run time comparison of training each offline RL algorithm (does not include evaluation costs). (Left) Breakdown of the cost of the added implementation details (mainly architecture changes) and the algorithmic details by each method. (Right) Total training time of each algorithm. While CQL and Fisher-BRC have significantly increased computational costs over their base online RL algorithm due to various added complexities (e.g. see Table 1), TD3+BC has effectively no increase. This results in less than half of the computational cost of these prior state-of-the-art algorithms.

Criticism to current offline RL algorithms

- 3. Instability of Trained Policies
  - Even if the average performance is reasonable, the agent may still *perform poorly on some episodes*.
  - This questions the empirical effectiveness of offline RL for *safety-critical real-world use cases* as well as the current trend of reporting *only the mean-value of the final policy* in offline benchmarking

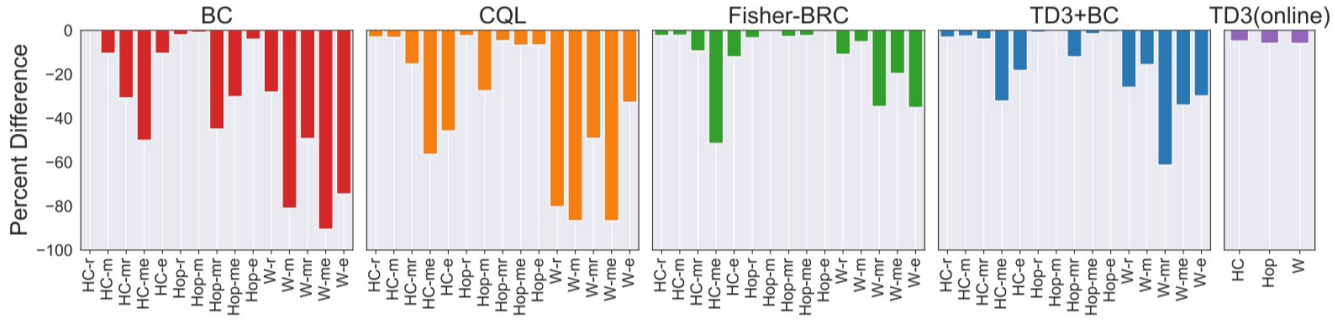


Figure 2: Percent difference of the worst episode during the 10 evaluation episodes at the last evaluation. This measures the deviations in performance at single point in time. HC = HalfCheetah, Hop = Hopper, W = Walker, r = random, m = medium, mr = medium-replay, me = medium-expert, e = expert. While online algorithms (TD3) typically have small episode variances per trained policy (as they should at *convergence*), all offline algorithms have surprisingly high episodic variances for *trained* policies.

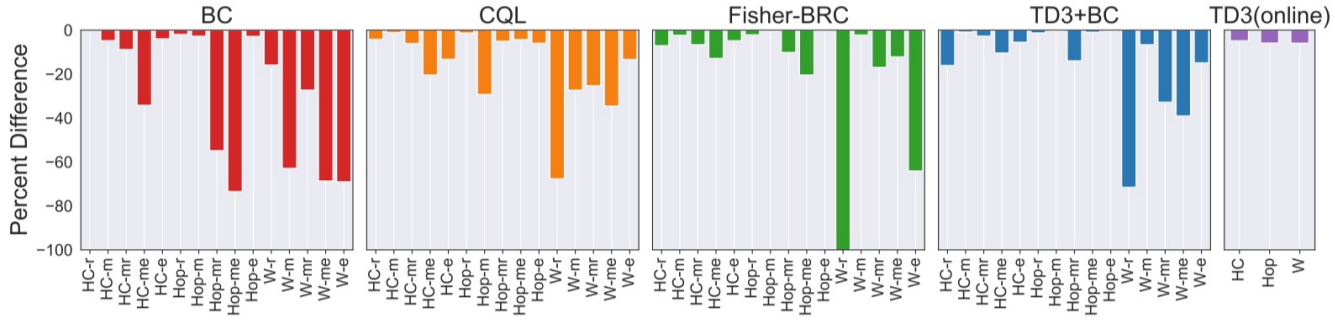


Figure 3: Percent difference of the worst evaluation during the last 10 evaluations. This measures the deviations in performance over a period of time. HC = HalfCheetah, Hop = Hopper, W = Walker, r = random, m = medium, mr = medium-replay, me = medium-expert, e = expert. Similarly to the result in Figure 2, all offline-trained policies have significant variances near the final stage of training that are absent in the online setting.

New algorithm: TD3 + BC

- 1. Add a behavior cloning regularization term to the standard policy update step of TD3, to push the policy towards favoring actions contained in the dataset D:

$$\pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s\sim\mathcal{D}}\left[Q(s,\pi(s))\right] \rightarrow \pi = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\lambda\, Q(s,\pi(s)) - (\pi(s)-a)^2\right]. \quad (3)$$

  - Given the dataset of *N* transitions (s_i, a_i), we define the scalar λ as:

$$\lambda = \frac{\alpha}{\frac{1}{N}\sum_{(s_i,a_i)} |Q(s_i,a_i)|}.$$

  - We observe that the balance between RL (in maximizing Q) and imitation (in minimizing the BC term), is highly susceptible to the scale of Q.
  - This formulation has the added benefit of normalizing the learning rate across tasks,

    as the gradient $\nabla_a Q(s,a)$ will also be dependent on the scale of Q. (use a = 2.5 in experiments.)

- 2. Normalize the features of every state in the provided dataset.

$$s_i = \frac{s_i - \mu_i}{\sigma_i + \epsilon},$$

(i th feature of the state s)

Experiments

- Performances on the D4RL benchmark of Open AI gym MuJoCo tasks

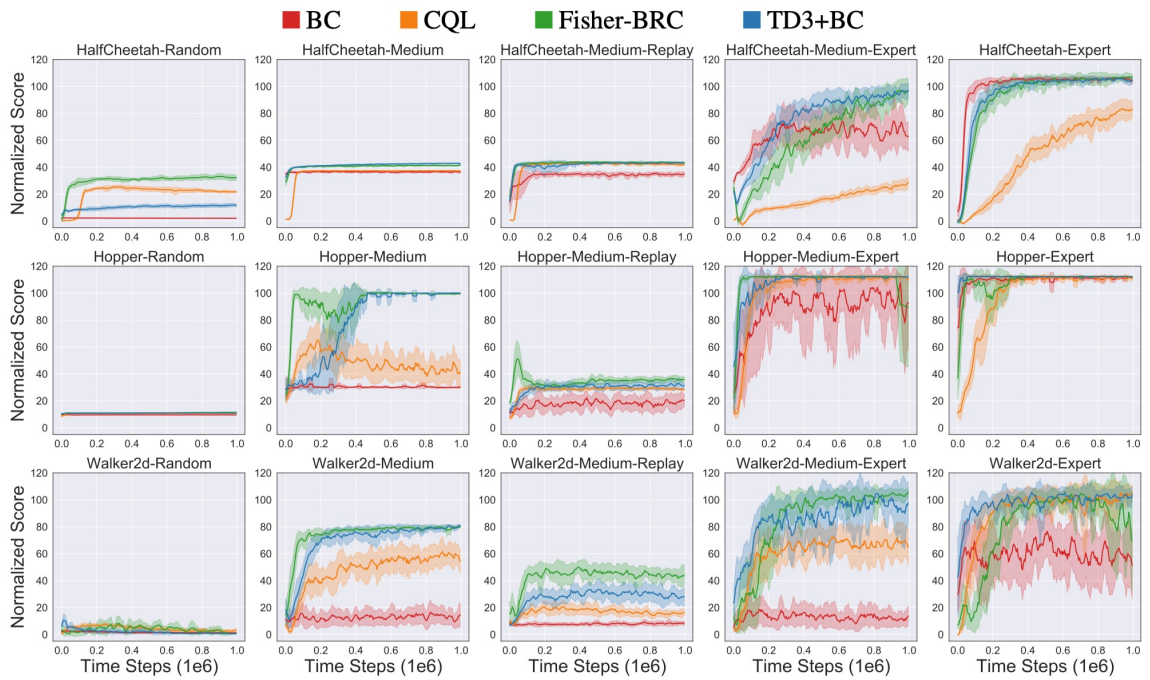


Figure 4: Learning curves comparing the performance of TD3+BC against offline RL baselines in the D4RL datasets. Curves are averaged over 5 seeds, with the shaded area representing the standard deviation across seeds. TD3+BC exhibits a similar learning speed and final performance as the state-of-the-art Fisher-BRC, without the need of pre-training a generative model.

Experiments

- Ablation study

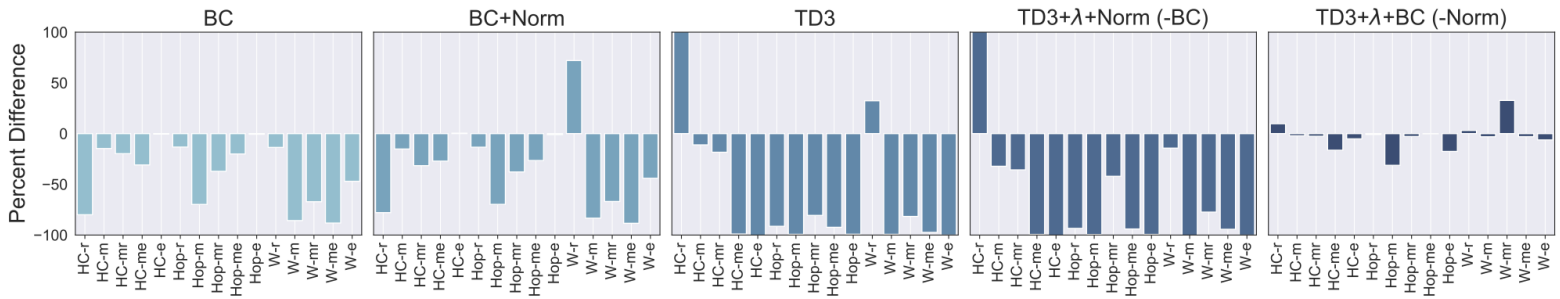


Figure 5: Percent difference of the performance of an ablation of our proposed approach, compared to the full algorithm. TD3+λ+BC+Norm refers to the complete algorithm, where Norm refers to the state feature normalization. HC = HalfCheetah, Hop = Hopper, W = Walker, r = random, m = medium, mr = medium-replay, me = medium-expert, e = expert. As expected, both BC and TD3 are necessary components to achieve a strong performance. While removing state normalization is not devastating to the performance of the algorithm, we remark it provides a boost in performance across many tasks, while being a straightforward addition.

- *Without behavior cloning regularization*, the RL algorithm alone is insufficient to achieve a high performance.

Experiments

- Hyperparameter α

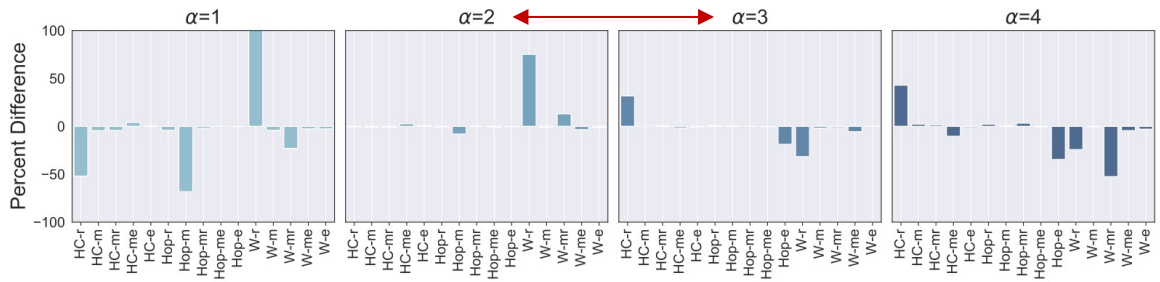
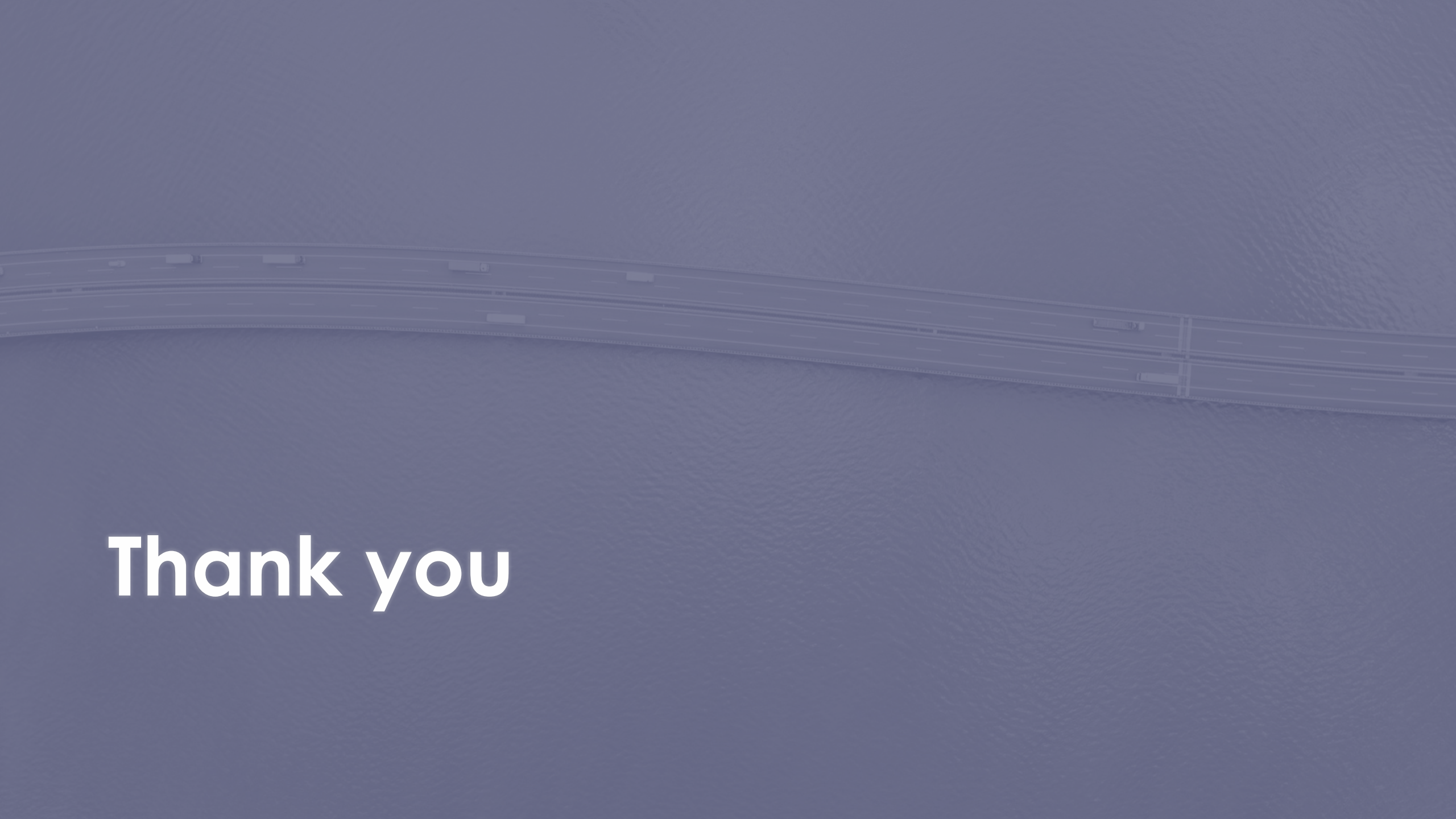


Figure 6: Percent difference of the performance of an ablation over α, compared to the full algorithm. Recall the form of the sole hyperparameter $\lambda = \frac{\alpha}{\sum |Q(s,a)|}$, where λ weights the Q function during the policy update. $\alpha = 2.5$ is used by default. HC = HalfCheetah, Hop = Hopper, W = Walker, r = random, m = medium, mr = medium-replay, me = medium-expert, e = expert. While there is effectively no performance difference between $\alpha = 2$ and $\alpha = 3$, we remark that performance begins to degrade on select tasks as the algorithm begins to approach more RL ($\alpha = 4$) or imitation ($\alpha = 1$).

Minimal code changes

```
135                 # Delayed policy updates
136                 if self.total_it % self.policy_freq == 0:
137
138                     # Compute actor loss
139                     pi = self.actor(state)
140                     Q = self.critic.Q1(state, pi)
141                     lmbda = self.alpha/Q.abs().mean().detach()
142
143                     actor_loss = -lmbda * Q.mean() + F.mse_loss(pi, action)
144
145                     # Optimize the actor
146                     self.actor_optimizer.zero_grad()
147                     actor_loss.backward()
148                     self.actor_optimizer.step()
```

https://github.com/sfujim/TD3_BC/blob/main/TD3_BC.py


Thank you