TARGET RATE OPTIMIZATION: AVOIDING ITERATIVE ERROR EXPLOITATION

Braham Snyder, Amy Zhang, Yuke Zhu
UT Austin
braham@utexas.edu

ABSTRACT

Many real-world reinforcement learning (RL) problems remain intractable. A key issue is that sample-efficient RL algorithms are unstable. Early stopping sometimes works around this. Yet early stopping in RL can be difficult, since the instability itself can result in few training steps having good policies. Standard early stopping stops all learning. Fixing the early stopping implicitly used with most target networks might be more robust. That is, in algorithms like DQN, the target update rate already early-stops DQN's target-fitting subproblems. Currently, practitioners must either hope the default target rate performs well, or tune it with an expensive grid search over online returns. Moreover, within a run, algorithms like DQN continue to update the target even when the updates *increase* the training error. This degrades value estimates, which degrades returns. Newer off-policy and offline RL algorithms lessen this well-known deadly triad divergence, but often require excessive pessimism to avoid it, gaining stability but at lower return. To combat these issues, we propose adding optimization of the training error w.r.t. the target update rate. Our algorithm, Target Rate Optimization, empirically prevents divergence and increases return by up to ~3× on a handful of discrete- and continuous-action RL problems.

1 INTRODUCTION

Immensely valuable models like ChatGPT are trained with unsupervised learning, supervised learning, and RL from human feedback (RLHF) (Knox & Stone, 2008; Christiano et al., 2017; Radford et al., 2018; Brown et al., 2020; Ouyang et al., 2022). However, almost all real-world RL so far is *bandit learning*, which only maximizes the immediate value per decision. Like supervised and unsupervised learning, bandit learning cannot stitch good decisions together in the way that RL can. RL's stitching enables it to learn policies far better than the best data in the dataset (Brandfonbrener et al., 2022; Kumar et al., 2022; Chebotar et al., 2023).

Although RL is the most promising approach for many problems, it is not often used in practice. This is partly because sample-efficient RL algorithms require three components that cause instability when combined. These three components are *the deadly triad*: (i) function approximation; (ii) training for an arbitrary target policy from arbitrary actions; and (iii) bootstrapping estimates from estimates at other states (Sutton & Barto, 2018; Tsitsiklis & Van Roy, 1996). Modern deadly triad algorithms help, yet do not fully solve instability (van Hasselt et al., 2018; Xiao et al., 2021; Schuurmans et al., 2021; Kumar et al., 2021a; Agarwal et al., 2019). Regularizing the learned policy more towards the dataset also helps stability, but often decreases peak returns. A partial workaround is one-step RL: estimate the behavior policy, then take just one policy improvement step. This prevents stitching value estimates beyond one step, but it surprisingly often beats algorithms that stitch (Brandfonbrener et al., 2021; Mathieu et al., 2023; Gulcehre et al., 2021; Goo & Niekum, 2022). A more standard partial workaround is early stopping. However, in the common case where online evaluations are costly, early stopping is unsettled and difficult (Kumar et al., 2021b). This is partly because offline hyperparameter tuning is unsettled and difficult (Paine et al., 2020; Zhang & Jiang, 2021). The difficulty of early stopping in offline RL is similarly why, for online RL, the number of optimization updates per environment step is either naively left at one, or tuned with an expensive grid search over online returns. Increasing this update-to-data (UTD) ratio often decreases returns, even though it is necessary for data efficiency (Li et al., 2023). Many algorithms have been pro-

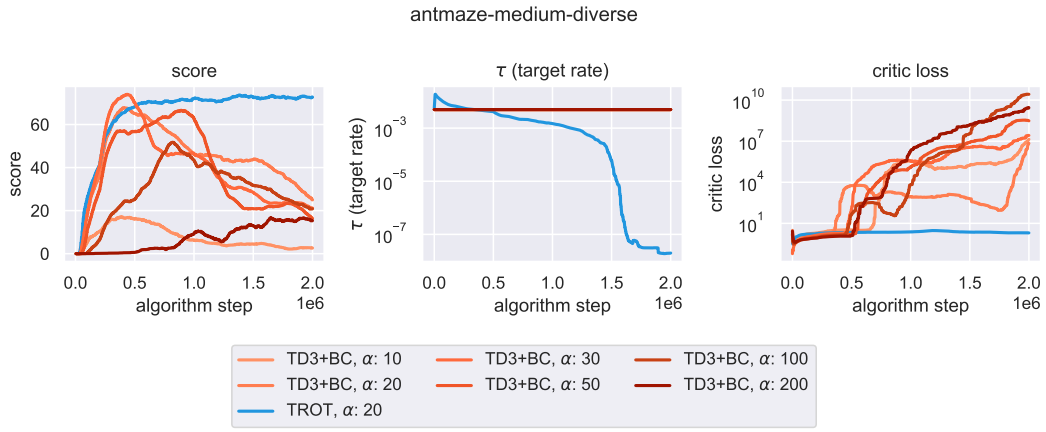

Figure 1: TD3+BC can diverge, scoring poorly even after tuning its pessimism setting, α. We show `antmaze-medium-diverse-v2` here. With zero changes other than its automatic target rate tuning, TROT gives TD3+BC stable high return. In the real world, we frequently cannot evaluate the score of every training run many times during training, which is why the standard in benchmarking is to compare the final return.

posed to fix these issues with more robust convergence guarantees than, e.g., DQN. However, these convergent algorithms have not been widely used. Some works claim this is due to their complexity (Xiao et al., 2021; Patterson et al., 2021).

Brandfonbrener et al. (2021) study issues with iterative off-policy evaluation, a component of standard off-policy and offline learning. They separate two sources of error: (a) iterative error exploitation and (b) distribution shift. Iterative error exploitation is the accumulation of error due to bootstrapping. That is, bootstrapping can amplify the errors from finite data and function approximation. Distribution shift, in off-policy RL, is the difference between the collected state-action distribution and the target policy's state-action distribution. Error due to distribution shift is well-studied both in machine learning, and in RL in particular (Chen & Jiang, 2019; Wang et al., 2020; Zanette, 2020; Fujimoto et al., 2018a; Wang et al., 2021). In this work, we focus on instability due to iterative error exploitation.

We take a simplistic and empirical approach to convergence. For further simplicity, we focus on offline RL, though our approach extends to online RL more naturally than alternatives we discuss. We propose two algorithms. Both of our algorithms interleave one step of standard, semi-gradient optimization for the critic with one step of optimization for the target update rate. One of our algorithms, TROT, incorporates full-gradient optimization. Fig. 1 shows how TROT can significantly increase TD3+BC's returns.

2 Preliminaries

We use the standard Markov decision process formulation (Sutton & Barto, 2018). $\mathcal{D}$ is an offline dataset (Levine et al., 2020) of transitions (s, a, r, s'), where s is a state, a is the chosen action in that state, and the environment gives both a reward $r = R(s, a)$ and a next state s' for that state-action pair. We aim to learn a policy π from states to actions that maximizes the expected cumulative rewards once deployed, discounted by $\gamma \in [0, 1)$ at each step for variance reduction: $\mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$. The expectation is over the stochasticity of the actions our policy takes, and of the environment's rewards and next states.

For the discrete-action setting, we aim to learn parameters θ for a state-action value function $Q_\theta(s, a)$ to match the optimal state-action value function, $Q^*(s, a)$. The latter gives the expected cumulative rewards of the optimal policy π^* if we first take action a in state s. That is, $Q^*(s, a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid (s_0, a_0) = (s, a),\, a_{1:\infty} \sim \pi^*\right]$. The expectation is again over the stochasticity of our policy, the rewards, and the next states. Q^* is defined this way so that

if we can learn $Q_\theta = Q^*$, the optimal policy is $\arg\max_a Q_\theta(s, a)$. Estimating the state-action value instead of only the state value avoids having to learn a transition model for the environment. Two ways to estimate Q^* are Fitted Q-Iteration (FQI) (Ernst et al., 2005; Riedmiller, 2005) and Q-learning (Watkins & Dayan, 1992). Currently, most deep off-policy RL algorithms are based on the Deep Q-Network (DQN) (Mnih et al., 2015), which combines FQI and Q-learning (Fu et al., 2019).

Similar to FQI and Q-learning, DQN learns Q^* using a Bellman optimality equation (Bellman, 1956), $Q^*(s, a) = r + \gamma \mathbb{E}_{s'}[\max_{a'} Q^*(s', a')]$. r and s' again come from the environment via (s, a), and all four are given for every transition in our offline dataset $\mathcal{D}$. Under certain assumptions, the unique Q that solves that equation must indeed be optimal, i.e., give the maximum cumulative reward starting from any state-action pair. One way to solve that equation is to minimize the squared difference between both sides parameterized by θ, giving the mean squared Bellman error (MSBE):

$$\mathcal{L}(\theta) := \mathop{\mathbb{E}}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q_\theta(s, a) - y \right)^2 \right]. \tag{1}$$

where $y := r + \gamma \max_{a'} Q_\theta(s', a')$ is a target toward which $Q_\theta(s, a)$ is trained. In DQN's case, this means putting a stop-gradient on that target. DQN also delays updates to its estimator of this target, similar to FQI. That is, the target y is instead estimated by a periodically updated copy $\bar{\theta}$ of the main parameters θ. When the target estimator is lagged in this way, and the value estimators Q_θ and $Q_{\bar{\theta}}$ are neural networks, the networks for target estimation $Q_{\bar{\theta}}$ are called *target networks* (Mnih et al., 2015). We give DQN pseudocode in Algorithm 1. Value learning algorithms like DQN that use stop-gradients are sometimes called *semi-gradient* algorithms, though treating the successor estimate as a fixed target is standard even outside gradient-based algorithms. None of these algorithms minimize the MSBE, Eq. (1). Rather, when they converge, they minimize the mean squared *projected* Bellman error (MSPBE). The MSPBE ignores error due to function approximation. With an overparameterized value function, there is zero approximation error, but the optimization and resulting solutions still differ for algorithms similar to DQN if one removes the stop-gradient (Xiao et al., 2021; Fujimoto et al., 2022). Indeed, removing the target network and stop-gradient gives a simple convergent algorithm, the *residual gradient* (RG) (Baird, 1995). That is, full-gradient descent of the MSBE, Eq. (1). Since gradient descent converges robustly, RG avoids deadly triad divergence. Unfortunately, RG estimates value poorly when data is missing (Fujimoto et al., 2022).

For benchmarking, we implement TRO on TD3+BC (Fujimoto & Gu, 2021), perhaps the most standard offline RL algorithm for continuous actions. To handle continuous actions, TD3+BC starts from a standard online RL algorithm: TD3 (Fujimoto et al., 2018b). To handle offline learning, TD3+BC uses the most common approach: add a behavior cloning (BC) term to the loss (Humphreys et al., 2022; Nakano et al., 2021). TD3+BC also normalizes the weight of the actor's RL term by the magnitude of the Q estimates in order to keep the RL weight and BC weight roughly equal.

3 Related Work

Early stopping the projection. As covered in Section 1, the standard, partial workaround for return degradation is early stopping of *all* learning, but it has drawbacks. One perspective is that updating the target too many times might be an important issue, since target updates are often the only time the training loss increases within a run. Instead of stopping all learning, one could stop the target updates, i.e., the projection. Fu et al. (2019) study this angle. Large divergence is rare in their experiments, likely because they study online RL. They early-stop the projection using oracle approaches. We speculate early-stopping the projection might lead to an even greater benefit in offline RL, where the risk of overfitting and iterative error exploitation seems greater because one cannot collect new data. They do not look for a practical algorithm. Additionally, early stopping cannot increase the target rate, which means it is not a natural fit for a soft target update. This is a significant drawback because a soft target update often outscores a hard target update. Moreover, the extension of an early-stopping approach to online RL is perhaps not clear, since it is not obvious how to handle the first target update step after collecting new data. Further, in our preliminary testing, our optimization algorithms (TRO) outscored early-stopping equivalents we developed. Those early-stopping equivalents similarly used only training losses. However, early-stopped projection is likely worth further study, including approaches with validation losses.

Convergence. Algorithms that treat their successor estimates as fixed targets, like DQN, Q-learning, and TD learning, provably follow no gradient in the general case. If they instead were gradient descent, convergence would be easier to study and ensure. One exception is when the environment is reversible (Ollivier, 2018; Brandfonbrener & Bruna, 2019; Liu & Olshevsky, 2020). Another exception is soft Q-learning, i.e., Q-learning with a smooth maximum, which can equal the entropy-regularized policy gradient (Schulman et al., 2017). Fellows et al. (2023) theorize convergence conditions for periodically updated targets like in DQN. They also include an experiment with a momentum term for the target updates, but they do not optimize the target update rate. There are also many other theoretical and empirical works showing target networks help convergence (van Hasselt et al., 2018; Zhang et al., 2021). To our knowledge, none optimize the target network update rate. One alternative to a target network is a different form of soft Q-learning (Asadi & Littman, 2017; Kim et al., 2019). The softness hyperparameter could be tuned similar to how our algorithm tunes the target update rate, but an advantage of target rate optimization is that target networks are more standard.

Provably convergent algorithms. Many algorithms have been developed for online RL that provably converge under more general conditions than, e.g., Q-learning. This includes TDC (Sutton et al., 2009), QRC (Ghiassian et al., 2020; Patterson et al., 2021), emphatic methods (Sutton et al., 2015; Mathieu et al., 2023), and Retrace (Munos et al., 2016). As we mentioned, existing convergent algorithms are not widespread, possibly because algorithms like DQN are simpler (Xiao et al., 2021). We instead take an empirical approach, making a minimal modification applicable to all DQN-like algorithms. Since DQN-like algorithms are more commonly used thus far, this might make it easier for practitioners and other researchers to fix divergence in existing code and algorithms.

Return degradation. Return degradation, the common pattern of a drop in rewards over the course of training, has been studied in both offline and online RL. In online RL, a drop in returns is sometimes called *policy collapse*, though it is unclear if the causes are the same. Such a drop can occur even with on-policy algorithms (Dohare et al., 2023). In off-policy RL, van Hasselt et al. (2015); Chen et al. (2021); Hiraoka et al. (2021) take the minimum of multiple value estimates to enable a higher UTD. This minimum counteracts overestimation due to maximization bias (Thrun & Schwartz, 1993). The UTD is also sometimes called the *replay ratio*. Solving overestimation in offline RL rather than online RL might be a simpler path to a general solution, since offline RL is the limit as UTD goes to infinity. Agarwal et al. (2019) show standard off-policy algorithms often score poorly in offline RL, because their return degrades. Brandfonbrener et al. (2021) demonstrate that even offline algorithms degrade unless they are strongly regularized towards the offline dataset, preventing maximum returns. Kumar et al. (2021a) propose the DR3 regularizer, improving returns and sometimes stability. However, they show DR3 does not completely fix return degradation.

4 Target Rate Optimization

As alluded to in Section 3, the standard off-policy and offline RL algorithms already incorporate a form of early stopping: the target update period. The target update period early-stops the projection after a fixed number of critic steps. Although this early stopping of the projection is different from early stopping of all learning, the target update rate is similarly a sensitive hyperparameter (Fu et al., 2019). The standard approach to tuning the target update rate, if it is tuned, is again an expensive grid search. A more efficient approach to setting the target rate might be to early-stop the projection with a validation split, as in supervised learning. However, this still has the drawbacks we discuss in Section 3. Yet early stopping is appealing, since it would automatically adjust the regularization strength over the course of training. In particular, it seems likely that later targets require stronger regularization as we approach a good value function. Possibly supporting this, one-step RL (Brandfonbrener et al., 2021) shows that zero off-policy target updates can greatly improve returns, and Li et al. (2023) find that adjusting the regularization technique online can improve performance in online RL. The latter might be due to the optimal regularization strength changing over the course of training. Moreover, as mentioned, the standard training loss frequently diverges, but naive minimization of training losses (like the residual gradient) are ineffective.

We combat all of these issues by adding optimization of training losses w.r.t. a soft target update rate, τ. That is, we optimize the rate of the exponential moving average (EMA) target introduced by DDPG (Lillicrap et al., 2015). In preliminary experiments, using a separate optimizer for τ worked

better than a combined optimizer with the main network. We interleave the optimizer steps one-to-one. We call the general form of our algorithm *Target Rate Optimization*. Algorithm 1 shows the pseudocode, added to offline DQN for simplicity. However, we suspect TRO could help any value estimator in the deadly triad setting with a target update rate. The most common deadly triad algorithms require such an estimator. We propose two algorithms, corresponding to two losses for the target rate, $\mathcal{L}(\tau)$. Our algorithms are **T**arget **R**ate **O**ptimization for the **T**arget net's MSBE (**TROT**), and **T**arget **R**ate **O**ptimization for the MS**P**BE (**TROP**). TROT optimizes τ using the MSBE of the target net with itself, whereas TROP optimizes τ using the standard DQN-like loss. That is, TROP uses the main net for the current state-action, and the target net for the next state-action. We keep the soft target update rate, τ, positive and near zero by defining $\tau := e^{\tilde{\tau}}$ and optimizing $\tilde{\tau} \in \mathbb{R}$. We omit this from the pseudocode for simplicity.

We write $Q_{\bar{\theta}}$ and $\bar{y}$ to emphasize the dependence of these two value estimates on the target parameters $\bar{\theta}$, and thus their dependence on τ and $\tilde{\tau}$ as well. We update the target parameters $\bar{\theta}$ at the start of every DQN step, rather than at the end of every DQN step, so that the TRO gradient is defined for the first iteration. Since the target parameters and main parameters are equal on the first DQN step, this is still exactly DQN (with a soft target update) if the TRO learning rate is set to zero. That is, both TROT and TROP recover the base algorithm if the TRO learning rate is set to zero.

Algorithm 1 Target Rate Optimization added to offline DQN. Changes from DQN are blue.

Given an offline dataset $\mathcal{D}$ of transitions $\{(s, a, r, s')\}$, and an initial soft target update rate, τ
Randomly initialize the parameters, θ, of an action-value function, Q
Initialize the target parameters $\bar{\theta} := \theta$
for each algorithm step **do**
  Update the target parameters $\bar{\theta} := \tau\theta + (1 - \tau)\bar{\theta}$
  Sample a batch $\{(s, a, r, s')\}$ from $\mathcal{D}$
  Compute the targets $\bar{y} := r + \max_{a'} Q_{\bar{\theta}}(s', a')$
  Take one critic gradient descent step following $\nabla_\theta(\bar{y} - Q_\theta(s, a))^2$
  Take one TRO gradient descent step following $\nabla_\tau \mathcal{L}(\tau)$, where

$$\mathcal{L}(\tau) := \begin{cases} (\bar{y} - Q_{\bar{\theta}}(s, a))^2 & \text{for TROT} \\ (\bar{y} - Q_\theta(s, a))^2 & \text{for TROP} \end{cases}$$

end for

TROT has an advantage over TROP in that the MSBE bounds the value error up to a constant (Williams & Baird, 1993), whereas the MSPBE gives no bound (Bertsekas & Tsitsiklis, 1996). As we noted, directly minimizing the MSBE (the residual gradient) gives poor value estimates if data is missing, but we find that minimizing the MSBE indirectly through TD updates fixes this. TROT aims to optimize the MSBE of the target net, not the main net, but we find the MSBE of the main net tends to follow the MSBE of the target net. One disadvantage of TROT is that stochastic environments and policies cause double sampling bias (Baird, 1995). We ignore double sampling bias, following prior work (Kostrikov et al., 2019; Nachum et al., 2019; Zhang et al., 2019; Cheng et al., 2022). To extend TROT to highly stochastic settings, there are many approaches to handle the bias (Antos et al., 2006; Geist et al., 2016; Dai et al., 2017; Zhu & Ying, 2019; Gimelfarb et al., 2021).

Small details in RL often have large effects (Kang et al., 2023; Henderson et al., 2017; Engstrom et al., 2020). For example, for a single batch, the order of the critic, actor, and α optimization steps in SAC can be significant (Tarasov et al., 2023b). For these reasons, we tested many such TRO design decisions in all combinations, for both TROT and TROP. None of those decisions affect plain TD3+BC or DQN, and we found most of the decisions had small effects. We briefly discuss three of those decisions in Appendix C.

5 Experiments

5.1 Design

We study offline RL for simplicity, but also because it suffers from iterative error exploitation, return degradation, and loss divergence even more than online RL. We design our experiments to determine whether TRO avoids these issues. First, as a proof of concept, we apply TRO on top

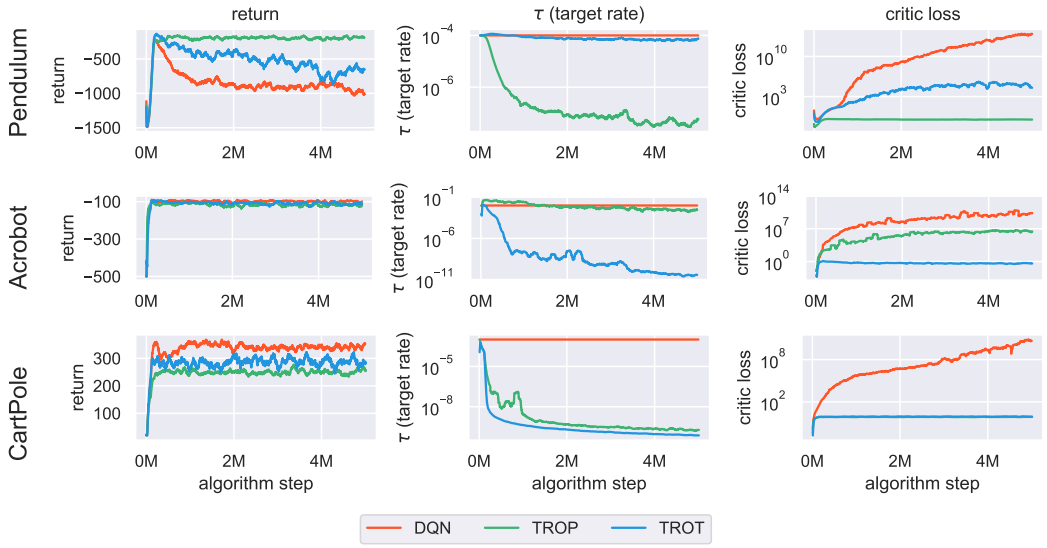

Figure 2: On the classic control tasks, offline DQN is known to diverge. Both TROP and TROT improve or eliminate divergence, typically with zero tuning of the TRO learning rate. *The only problem where we attempted tuning the TRO learning rate was Pendulum*, where we tuned only ■ TROT. We use 1e-6 there. Although DQN beats TRO on CartPole, we strongly suspect a TRO learning rate other than our default would match DQN's return there while retaining TRO's convergence. We average over 20 seeds.

of DQN, and follow Xiao et al. (2021) in using one of the simplest problem sets where standard deep RL algorithms are known to diverge. TRO improves or eliminates degradation and divergence. Second, to validate it helps DQN on a nontrivial problem, we also test it on an Atari game, where we again find it eliminates return degradation. Finally, we test if TROT can fix the return degradation of an offline RL algorithm used in practice, finding TROT gives up to ~3× higher final return.

5.2 Classic Control

To confirm TRO works at all, we test the classic control tasks (Brockman et al., 2016) using random offline data, where the simplest deep RL algorithms are known to diverge, and their return is known to degrade (Schuurmans et al., 2021; Xiao et al., 2021). For additional simplicity, we create a discrete-action version of the Pendulum setup from Xiao et al. (2021). As expected, DQN's loss divergence and return degradation persist despite hyperparameter tuning, detailed in Appendix E. We exclude their Reacher setup since it seems less discriminative than the other tasks. We follow their work in collecting one thousand random actions for Pendulum, and ten thousand random actions for CartPole and Acrobot. Like their work, for simplicity, we do not add any pessimism for any of our algorithms on these datasets. This might be especially reasonable for these datasets, since they are random actions.

For all classic control experiments except TROT on Pendulum, we did not tune the TRO learning rate. That is, we used the default 3e-4. We tuned only the critic learning rate and τ for both DQN and TRO, running for 500k algorithm steps (not shown). We then ran all the best configurations for 5M steps, and show in Fig. 2 only the single best configuration for each algorithm and dataset by final return. We again use a soft target update for our DQN baseline. For DQN, we also tuned the regular hard target updates (not shown) on all environments, which was reliably no better.

5.3 Atari

In initial experiments, we found that TROT often outscored TROP. As mentioned, TROT may also have an advantage in that the MSBE bounds the value error. Accordingly, the remainder of our results are for TROT alone, though we believe TROP is also worth further study. For a more complex

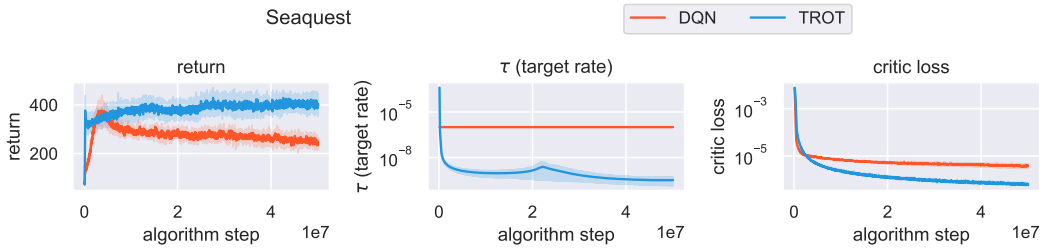

Figure 3: TROT remains stable on Seaquest for at least 50M algorithm steps, even *without tuning* the TRO learning rate. Although TROT performs best here with a high initial τ (shown), a high initial τ is not necessary for TROT to outscore even DQN's best hyperparameter configuration out of 20. The small peak in TROT's learned τ halfway through training may suggest a more cohesive optimization would do even better. We discuss that in Section 6. Shaded areas give standard deviations.

proof of concept than the classic control problems, we use offline DQN on 100k random actions from Seaquest (Bellemare et al., 2013; Machado et al., 2018). Similar to the DQN experiments in Kumar et al. (2021a); Agarwal et al. (2019), we confirm that DQN by itself degrades. We again did not tune the TRO learning rate at all, yet Fig. 3 shows TROT eliminates return degradation. We use the deterministic version of Seaquest to focus on return degradation, i.e., to ignore double sampling bias again.

5.4 Antmaze

For modern algorithms, the benchmark that most commonly causes divergence may be offline Antmaze (Fu et al., 2020). Since TD3+BC is possibly the most standard offline algorithm for continuous actions, we implement TROT on TD3+BC. TROT needs the BC component from TD3+BC because, to our knowledge, TRO is only pessimistic to the extent it avoids iterative error exploitation. To handle distribution shift, TRO requires additional pessimism, similar to one-step RL's additional pessimism. *We do not change TD3+BC's hyperparameters* aside from its pessimism, α, which we tune equally for both TROT and TD3+BC by itself. For TROT, we additionally tune our TRO learning rate.

Following prior work, we tune those two hyperparameters in combination and show the best score per dataset (An et al., 2021; Tarasov et al., 2023a). Tuning with limited online returns is common, since, as mentioned, offline hyperparameter tuning is unsettled and difficult. Moreover, tuning with limited online returns is often a realistic setup (Brandfonbrener et al., 2021; Kurenkov & Kolesnikov, 2021). To our knowledge, the pessimism hyperparameter for every existing offline algorithm is sensitive (Tarasov et al., 2022; Kang et al., 2023). TRO only slightly decreases the sensitivity of TD3+BC's pessimism, and this is likely to hold for other base offline algorithms. Offline algorithms without a sensitive pessimism hyperparameter are an active area of research (Hong et al., 2022). Such algorithms will probably still need mechanisms like TRO to handle iterative error exploitation.

Tuning a second hyperparameter for plain TD3+BC would be a fairer comparison against TD3+BC with a tuned TRO learning rate. However, in Section 5.2 and Section 5.3, TRO increased stability and final scores even without tuning the TRO learning rate. Similarly, Tarasov et al. (2023a) show that TD3+BC requires extensive changes to score highly on Antmaze: a larger discount factor, deeper neural networks, layer norm (Ba et al., 2016), larger batches, and reward scaling. Further, these changes do not eliminate return degradation and divergence on other datasets.

Baselines. As mentioned, details in RL often have large effects (Kang et al., 2023; Henderson et al., 2017; Engstrom et al., 2020). To find which details help, we use only controlled experiments for our benchmarking. As noted in Section 2, we focus on an instance of the most ubiquitous approach to offline RL. That is, we compare TD3+BC against itself with one change at a time. Other algorithms also suffer from the divergence on Antmaze shown in Fig. 1, so we expect our findings to generalize. IQL is one exception to that divergence (Kang et al., 2023), which could be due to IQL's learning rate decay for its actor. TRO's automatic τ schedule might be a more principled alternative to IQL's cosine learning rate decay, which: (i) does not react to changes during training; (ii) does not react

Table 1: Tarasov et al. (2023a) show their TD3+BC implementation ("ReBRAC's TD3+BC" in our table) performs poorly on Antmaze even when the pessimism is tuned per dataset. We similarly find low scores with a standard implementation. However, by optimizing τ, changing nothing else from TD3+BC, TROT greatly increases the scores on every dataset where plain TD3+BC degrades. 2M critic gradient steps, averaged over 10 seeds. The highest score per dataset is highlighted.

Dataset	ReBRAC's TD3+BC	TD3+BC	TROT
antmaze-umaze	66.3 ± 6.2	94.1 ± 7.2	92.3 ± 7.5
antmaze-umaze-diverse	53.8 ± 8.5	50.3 ± 14.8	50.5 ± 15.2
antmaze-medium-play	26.5 ± 18.4	56.0 ± 39.9	78.2 ± 13.9
antmaze-medium-diverse	25.9 ± 15.3	25.9 ± 36.7	71.5 ± 14.0
antmaze-large-play	0.0 ± 0.0	10.1 ± 13.0	18.8 ± 15.6
antmaze-large-diverse	0.0 ± 0.0	21.1 ± 14.2	23.1 ± 15.5

to changes to the environment and dataset; (iii) does not naturally fit online RL; and (iv) does not imply additional, more adaptive algorithms. Expanding on (iv): a hyperparameter-free (Cutkosky et al., 2023) algorithm similar to TRO might be worth pursuing. Despite these reasons to prefer TRO over cosine actor decay, we plan to compare against TD3+BC with that decay in future work. Underscoring the importance of such controlled experiments, Liu et al. (2023) find their algorithm built on SAC scores *over two times higher* on Antmaze than the same algorithm built on TD3. Similar to SAC, IQL uses learnable noise for its actor. Despite those disadvantages, TROT enables TD3+BC to beat half of the Antmaze scores for IQL in Tarasov et al. (2023a), who similarly tune IQL's two hyperparameters per dataset. However, because we focus on controlled comparisons, we do not include those IQL scores here. Likewise, we do not include CQL's scores, which additionally benefit from deeper nets (Tarasov et al., 2023a) and learning rate tuning. CQL also suffers return degradation, which led Kumar et al. (2021a) to develop DR3 for improved stability. We plan to test DR3 with TD3+BC for a controlled comparison against TRO.

Results. Within 2M algorithm steps, the Antmaze datasets TD3+BC degrades on are `antmaze-medium-play`, `antmaze-medium-diverse`, and `antmaze-large-play`. Table 1 shows TROT prevents that return degradation in every case. To check whether more training may cause additional degradation, we also trained TD3+BC for 5M algorithm steps on `antmaze-umaze`. As shown in Table 2, this causes TD3+BC to degrade there as well. In contrast, TROT again retains its peak returns. We speculate more training would cause degradation on other datasets, too.

5.5 Discussion

TRO requires extra gradient computations. In our experiments, we used one TRO training step per main training step, doubling the length per training run. On some problems, TRO's increased stability and decreased initial target rate sensitivity is likely worth this cost. We also speculate less frequent TRO steps would retain TRO's benefits while greatly reducing its costs.

TRO often improves over the base algorithm in a wide range of TRO learning rates. Tuning over $\{$1e-3, 1e-4, 1e-5, 1e-6$\}$ is usually enough. For example, we required only 1e-3 and 1e-4 for our Antmaze results, and in many other cases we did not attempt anything other than the default 3e-4.

When TROT does not improve over the base algorithm, it often appears to be due to short-sighted or otherwise ineffective optimization. That is, TROT does not seem to encounter the failure of the residual gradient on certain problems, where the residual gradient's return is always bad even if the MSBE is low (Fujimoto et al., 2022; Schuurmans et al., 2021). On runs where TROT succeeds at minimizing the MSBE of the target net more than the base algorithm would have, TROT typically outscores the base algorithm. Conversely, when TROT's MSBE is higher than the MSBE the base algorithm gets, the base algorithm typically outscores TROT. In Appendix B, we elaborate on this and discuss datasets that TRO does not help for. TRO can also cause a cyclical tau, as in Fig. 3, which we speculate is suboptimal.

Table 2: We trained TD3+BC for more steps on `antmaze-umaze`. This causes returns to degrade there, too. TROT again avoids degradation. 5M critic gradient steps, averaged over 5 seeds.

Dataset	TD3+BC	TROT
antmaze-umaze	66.2 ± 36.9	96.4 ± 6.0

6 Conclusion

TRO tends to stabilize the base algorithm as long as its added optimization succeeds, as discussed in Section 5.5. Consequently, the most promising follow-up might be similar algorithms with more cohesive optimization. *Residual algorithms*, weighted averages of the residual gradient and semi-gradient (Baird, 1995), might be a worthwhile baseline for that. However, their residual weight hyperparameter is sensitive (Zhang et al., 2019). Another approach that might optimize more cohesively is early stopping of the projection, though we list downsides in Section 3.

It is perhaps unclear whether uncertainty-based pessimism (Buckman et al., 2020) like SAC-N (An et al., 2021) ought to handle iterative error exploitation on its own, without algorithms like TROT. Yet, residual algorithms can decrease the number of critics SAC-N requires by more than an order of magnitude (Snyder & Zhu, 2022), which suggests such approaches may be important for uncertainty-based pessimism as well.

Given TROT's simplicity and effectiveness, it might be worth testing on additional problems. For instance, AlphaStar Unplugged (Mathieu et al., 2023) finds that degradation of their win-rate prevents iterative methods from outscoring one-step RL on a difficult real-time strategy game.

Acknowledgments

For helpful discussions and code, we thank Denis Tarasov, Chenjun Xiao, anonymous reviewers, Nan Jiang, Gergely Neu, Bo Dai, and RPL at UT Austin. We similarly thank the Weights and Biases support and engineering teams.

We also thank the authors of the libraries we use: cleanrl (Huang et al., 2022), CORL (Tarasov et al., 2022), d3rlpy (Seno & Imai, 2023), Python (Van Rossum et al., 1995), PyTorch (Paszke et al., 2019), NumPy and SciPy (Harris et al., 2020), matplotlib (Hunter, 2007), seaborn (Waskom, 2021), pandas (McKinney et al., 2011), MuJoCo (Todorov et al., 2012), and Stella (Mott et al., 1995).

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. *arXiv*, July 2019. doi: 10.48550/arXiv.1907.04543. 1, 4, 7

Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-Based Offline Reinforcement Learning with Diversified Q-Ensemble. *arXiv*, October 2021. doi: 10.48550/arXiv.2110.01548. 7, 9

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. In *Learning Theory: 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006. Proceedings 19*, pp. 574–588. Springer, 2006. 5

Kavosh Asadi and Michael L. Littman. An Alternative Softmax Operator for Reinforcement Learning. In *International Conference on Machine Learning*, pp. 243–252. PMLR, July 2017. URL `https://proceedings.mlr.press/v70/asadi17a.html`. 4

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv*, July 2016. doi: 10.48550/arXiv.1607.06450. 7

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pp. 30–37. Elsevier, 1995. 3, 5, 9

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013. 7

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956. 3

Frederik Benzing. Gradient Descent on Neurons and its Link to Approximate Second-Order Optimization. *arXiv*, January 2022. doi: 10.48550/arXiv.2201.12250. 16

Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996. 5

David Brandfonbrener and Joan Bruna. Geometric Insights into the Convergence of Nonlinear TD Learning. *arXiv*, May 2019. doi: 10.48550/arXiv.1905.12185. 4

David Brandfonbrener, William F. Whitney, Rajesh Ranganath, and Joan Bruna. Offline RL Without Off-Policy Evaluation. *arXiv*, June 2021. doi: 10.48550/arXiv.2106.08909. 1, 2, 4, 7, 16

David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? *arXiv*, June 2022. doi: 10.48550/arXiv.2206.01079. 1

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv*, June 2016. doi: 10.48550/arXiv.1606.01540. 6

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv*, May 2020. doi: 10.48550/arXiv.2005.14165. 1

Jacob Buckman, Carles Gelada, and Marc G. Bellemare. The Importance of Pessimism in Fixed-Dataset Policy Optimization. *arXiv*, September 2020. doi: 10.48550/arXiv.2009.06799. 9

Yevgen Chebotar, Quan Vuong, Alex Irpan, Karol Hausman, Fei Xia, Yao Lu, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. *arXiv preprint arXiv:2309.10150*, 2023. 1

Jinglin Chen and Nan Jiang. Information-Theoretic Considerations in Batch Reinforcement Learning. *arXiv*, May 2019. doi: 10.48550/arXiv.1905.00360. 2

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. *arXiv*, January 2021. doi: 10.48550/arXiv.2101.05982. 4

Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially Trained Actor Critic for Offline Reinforcement Learning. *arXiv*, February 2022. doi: 10.48550/arXiv.2202.02446. 5

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. *Advances in Neural Information Processing Systems*, 30, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html`. 1

Ashok Cutkosky, Aaron Defazio, and Harsh Mehta. Mechanic: A Learning Rate Tuner. *arXiv*, May 2023. doi: 10.48550/arXiv.2306.00144. 8

Will Dabney and Philip S. Thomas. Natural temporal difference learning. In *AAAI Conference on Artificial Intelligence*, 2014. URL `https://api.semanticscholar.org/CorpusID:10065534`. 16

Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation. *arXiv*, December 2017. doi: 10.48550/arXiv.1712.10285. 5, 16

Shibhansh Dohare, Qingfeng Lan, and A. Rupam Mahmood. Overcoming Policy Collapse in Deep Reinforcement Learning. *OpenReview*, May 2023. URL https://openreview.net/forum?id=m9Jfdz4ymO. 4

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. *arXiv*, May 2020. doi: 10.48550/arXiv.2005.12729. 5, 7

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005. 3

Mattie Fellows, Matthew J. A. Smith, and Shimon Whiteson. Why Target Networks Stabilise Temporal Difference Methods. *arXiv*, February 2023. doi: 10.48550/arXiv.2302.12537. 4

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing Bottlenecks in Deep Q-learning Algorithms. *arXiv*, February 2019. doi: 10.48550/arXiv.1902.10250. 3, 4

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. 7, 16, 18

Scott Fujimoto and Shixiang Shane Gu. A Minimalist Approach to Offline Reinforcement Learning. *arXiv*, June 2021. doi: 10.48550/arXiv.2106.06860. 3, 18

Scott Fujimoto, David Meger, and Doina Precup. Off-Policy Deep Reinforcement Learning without Exploration. *arXiv*, December 2018a. doi: 10.48550/arXiv.1812.02900. 2

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv*, February 2018b. doi: 10.48550/arXiv.1802.09477. 3

Scott Fujimoto, David Meger, Doina Precup, Ofir Nachum, and Shixiang Shane Gu. Why should i trust you, bellman? the bellman error is a poor replacement for value error. *arXiv*, January 2022. doi: 10.48550/arXiv.2201.12417. 3, 8, 16

Matthieu Geist, Bilal Piot, and Olivier Pietquin. Is the Bellman residual a bad proxy? *arXiv*, June 2016. doi: 10.48550/arXiv.1606.07636. 5

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient Temporal-Difference Learning with Regularized Corrections. In *International Conference on Machine Learning*, pp. 3524–3534. PMLR, November 2020. URL https://proceedings.mlr.press/v119/ghiassian20a.html. 4

Michael Gimelfarb, André Barreto, Scott Sanner, and Chi-Guhn Lee. Risk-Aware Transfer in Reinforcement Learning using Successor Features. *arXiv*, May 2021. doi: 10.48550/arXiv.2105.14127. 5

Wonjoon Goo and Scott Niekum. You Only Evaluate Once: a Simple Baseline Algorithm for Offline RL. In *Conference on Robot Learning*, pp. 1543–1553. PMLR, January 2022. URL https://proceedings.mlr.press/v164/goo22a.html. 1

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*, March 2021. doi: 10.48550/arXiv.2103.09575. 1

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019. 16

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. 9

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *arXiv*, September 2017. doi: 10.48550/arXiv.1709.06560. 5, 7

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout Q-Functions for Doubly Efficient Reinforcement Learning. *arXiv*, October 2021. doi: 10.48550/arXiv.2110.02034. 4

Joey Hong, Aviral Kumar, and Sergey Levine. Confidence-Conditioned Value Functions for Offline Reinforcement Learning. *arXiv*, December 2022. doi: 10.48550/arXiv.2212.04607. 7

Shengyi Huang, Rousslan Fernand JulienDossa Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João GM Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *The Journal of Machine Learning Research*, 23(1):12585–12602, 2022. 9

Peter C. Humphreys, David Raposo, Toby Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Alex Goldin, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. *arXiv*, February 2022. doi: 10.48550/arXiv.2202.08137. 3

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9 (03):90–95, 2007. 9

Nan Jiang. On Value Functions and the Agent-Environment Boundary. *arXiv*, May 2019. doi: 10.48550/arXiv.1905.13341. 16

Nan Jiang. Nan Jiang on X, June 2022. URL `https://twitter.com/nanjiang_cs/status/1535723582663565312`. 16

Bingyi Kang, Xiao Ma, Yirui Wang, Yang Yue, and Shuicheng Yan. Improving and Benchmarking Offline Reinforcement Learning Algorithms. *arXiv*, June 2023. doi: 10.48550/arXiv.2306.00972. 5, 7

Seungchan Kim, Kavosh Asadi, Michael L. Littman, and George Dimitri Konidaris. Deepmellow: Removing the need for a target network in deep q-learning. In *International Joint Conference on Artificial Intelligence*, 2019. URL `https://api.semanticscholar.org/CorpusID:197638361`. 4

W Bradley Knox and Peter Stone. Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE international conference on development and learning*, pp. 292–297. IEEE, 2008. 1

Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation Learning via Off-Policy Distribution Matching. *arXiv*, December 2019. doi: 10.48550/arXiv.1912.05032. 5

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline Reinforcement Learning with Implicit Q-Learning. *arXiv*, October 2021. doi: 10.48550/arXiv.2110.06169. 18

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization. *arXiv*, December 2021a. doi: 10.48550/arXiv.2112.04716. 1, 4, 7, 8

Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A Workflow for Offline Model-Free Robotic Reinforcement Learning. *arXiv*, September 2021b. doi: 10.48550/arXiv.2109.10813. 1

Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline Q-Learning on Diverse Multi-Task Data Both Scales And Generalizes. *arXiv*, November 2022. doi: 10.48550/arXiv.2211.15144. 1

Vladislav Kurenkov and Sergey Kolesnikov. Showing Your Offline Reinforcement Learning Work: Online Evaluation Budget Matters. *arXiv*, October 2021. doi: 10.48550/arXiv.2110.04156. 7

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv*, May 2020. doi: 10.48550/arXiv.2005.01643. 2

Qiyang Li, Aviral Kumar, Ilya Kostrikov, and Sergey Levine. Efficient Deep Reinforcement Learning Requires Regulating Overfitting. *arXiv*, April 2023. doi: 10.48550/arXiv.2304.10466. 1, 4

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv*, September 2015. doi: 10.48550/arXiv.1509.02971. 4

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv*, August 2019. doi: 10.48550/arXiv.1908.03265. 17

Rui Liu and Alex Olshevsky. Temporal Difference Learning as Gradient Splitting. *arXiv*, October 2020. doi: 10.48550/arXiv.2010.14657. 4

Yao Liu, Pratik Chaudhari, and Rasool Fakoor. Budgeting Counterfactual for Offline RL. *arXiv*, July 2023. doi: 10.48550/arXiv.2307.06328. 8

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018. 7

James Martens and Roger Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *arXiv*, March 2015. doi: 10.48550/arXiv.1503.05671. 16

Michaël Mathieu, Sherjil Ozair, Srivatsan Srinivasan, Caglar Gulcehre, Shangtong Zhang, Ray Jiang, Tom Le Paine, Richard Powell, Konrad Żołna, Julian Schrittwieser, David Choi, Petko Georgiev, Daniel Toyama, Aja Huang, Roman Ring, Igor Babuschkin, Timo Ewalds, Mahyar Bordbar, Sarah Henderson, Sergio Gómez Colmenarejo, Aäron van den Oord, Wojciech Marian Czarnecki, Nando de Freitas, and Oriol Vinyals. AlphaStar Unplugged: Large-Scale Offline Reinforcement Learning. *arXiv*, August 2023. doi: 10.48550/arXiv.2308.03526. 1, 4, 9

Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011. 9

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 3

Bradford Mott et al. Stella: "A Multi-Platform Atari 2600 VCS Emulator", 1995. URL `https://stella-emu.github.io/credits.html`. 9

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. *arXiv*, June 2016. doi: 10.48550/arXiv.1606.02647. 4

Ofir Nachum and Bo Dai. Reinforcement Learning via Fenchel-Rockafellar Duality. *arXiv*, January 2020. doi: 10.48550/arXiv.2001.01866. 16

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Trust-PCL: An Off-Policy Trust Region Method for Continuous Control. *arXiv*, July 2017. doi: 10.48550/arXiv.1707.01891. 16

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and D. Schuurmans. AlgaeDICE: Policy Gradient from Arbitrary Experience. *arXiv preprint arXiv:1906.03963*, 2019. URL `https://arxiv.org/abs/1906.03963`. 5

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback. *arXiv*, December 2021. doi: 10.48550/arXiv.2112.09332. 3

Yann Ollivier. Approximate Temporal Difference Learning is a Gradient Descent for Reversible Policies. *arXiv*, May 2018. doi: 10.48550/arXiv.1805.00869. 4

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022. URL https://api.semanticscholar.org/CorpusID:246426909. 1

Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter Selection for Offline Reinforcement Learning. *arXiv*, July 2020. doi: 10.48550/arXiv.2007.09055. 1

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 9

Andrew Patterson, Adam White, and Martha White. A Generalized Projected Bellman Error for Off-policy Value Estimation in Reinforcement Learning. *arXiv*, April 2021. doi: 10.48550/arXiv.2104.13844. 2, 4

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning. *arXiv*, April 2023. doi: 10.48550/arXiv.2304.01315. 18

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018. 1

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. 16

Martin A. Riedmiller. Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, 2005. URL https://api.semanticscholar.org/CorpusID:6921329. 3

John Schulman, Xi Chen, and Pieter Abbeel. Equivalence Between Policy Gradients and Soft Q-Learning. *arXiv*, April 2017. doi: 10.48550/arXiv.1704.06440. 4

Dale Schuurmans, Chenjun Xiao, Bo Dai, Jincheng Mei, Oscar A. Ramirez, Ramki Gummadi, and Chris Harris. Understanding Deep Value Estimation, December 2021. URL https://slideslive.com/38970598/understanding-deep-value-estimation. 1, 6, 8

Takuma Seno and Michita Imai. d3rlpy, September 2023. URL https://github.com/takuseno/d3rlpy. 9

Braham Snyder and Yuke Zhu. Raisin: Residual Algorithms for Versatile Offline Reinforcement Learning. *OpenReview*, September 2022. URL https://openreview.net/forum?id=d1oQqDvB7GQ. 9

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 1, 2

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th annual international conference on machine learning*, pp. 993–1000, 2009. 4

Richard S. Sutton, A. Rupam Mahmood, and Martha White. An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning. *arXiv*, March 2015. doi: 10.48550/arXiv.1503.04269. 4

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. CORL: Research-oriented Deep Offline Reinforcement Learning Library. *arXiv*, October 2022. doi: 10.48550/arXiv.2210.07105. 7, 9, 18

Denis Tarasov, Vladislav Kurenkov, Alexander Nikulin, and Sergey Kolesnikov. Revisiting the Minimalist Approach to Offline Reinforcement Learning. *ArXiv*, 2023a. URL https://www.semanticscholar.org/paper/Revisiting-the-Minimalist-Approach-to-Offline-Tarasov-Kurenkov/d50ddfa3f02b8fa5d0a7bfd396ddaf9afc895cb1. 7, 8, 18

Denis Tarasov, Alexander Nikulin, Dmitry Akimov, Vladislav Kurenkov, and Sergey Kolesnikov. Corl/algorithms/offline/sac_n.py at 6afec90484bbf47dee05fdf525e26a3ebe028e9b · tinkoff-ai/corl, 9 2023b. URL https://github.com/tinkoff-ai/CORL/blob/6afec90484bbf47dee05fdf525e26a3ebe028e9b/algorithms/offline/sac{_}n.py{#}L381. 5

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school*, pp. 255–263, 1993. 4

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012. 9

John Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, 9, 1996. 1

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv*, September 2015. doi: 10.48550/arXiv.1509.06461. 4

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep Reinforcement Learning and the Deadly Triad. *arXiv*, December 2018. doi: 10.48550/arXiv.1812.02648. 1, 4

Guido Van Rossum, Fred L Drake, et al. *Python reference manual*, volume 111. Centrum voor Wiskunde en Informatica Amsterdam, 1995. 9

Ruosong Wang, Dean P. Foster, and Sham M. Kakade. What are the Statistical Limits of Offline RL with Linear Function Approximation? *arXiv*, October 2020. doi: 10.48550/arXiv.2010.11895. 2

Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham M. Kakade. Instabilities of Offline RL with Pre-Trained Neural Representation. *arXiv*, March 2021. doi: 10.48550/arXiv.2103.04947. 2

Michael L Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6 (60):3021, 2021. 9

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 3

Ronald J Williams and Leemon C Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Tech. rep. NU-CCS-93-14, Northeastern University, College of Computer . . . , 1993. 5

Chenjun Xiao, Bo Dai, Jincheng Mei, Oscar A. Ramirez, Ramki Gummadi, Chris Harris, and Dale Schuurmans. Understanding and Leveraging Overparameterization in Recursive Value Estimation. *ICLR*, October 2021. URL https://openreview.net/forum?id=shbAgEsk3qM. 1, 2, 3, 4, 6, 16

Andrea Zanette. Exponential Lower Bounds for Batch Reinforcement Learning: Batch RL can be Exponentially Harder than Online RL. *arXiv*, December 2020. doi: 10.48550/arXiv.2012.08005. 2

Shangtong Zhang, Wendelin Boehmer, and Shimon Whiteson. Deep Residual Reinforcement Learning. *arXiv*, May 2019. doi: 10.48550/arXiv.1905.01072. 5, 9

Shangtong Zhang, Hengshuai Yao, and Shimon Whiteson. Breaking the Deadly Triad with a Target Network. *arXiv*, January 2021. doi: 10.48550/arXiv.2101.08862. 4

Siyuan Zhang and Nan Jiang. Towards Hyperparameter-free Policy Selection for Offline Reinforcement Learning. *arXiv*, October 2021. doi: 10.48550/arXiv.2110.14000. 1

Yuhua Zhu and Lexing Ying. Borrowing From the Future: An Attempt to Address Double Sampling. *arXiv*, December 2019. doi: 10.48550/arXiv.1912.00304. 5

A Residual Gradient Experiments

On our new, discrete-action version of the offline Pendulum problem from random actions (Xiao et al., 2021), we find RG still scores significantly lower than the semi-gradient (SG), even when RG is given a deterministic start state. RG's underperformance persisted despite tuning the learning rate and batch size. This provides initial evidence against the boundary conjecture in Appendix D of Jiang (2019), i.e., that RG fails due to stochastic start states. That said, Jiang (2022) instead conjectures that SG might outperform RG by implicitly handling difficult details as stochasticity. This may connect with the claim that RG is not invariant (Dabney & Thomas, 2014). K-FAC, which may (Martens & Grosse, 2015) or may not (Benzing, 2022) be closer to the natural gradient than Adam is, did not help much in our experiments.

Larger batch and dataset sizes did improve RG a bit, at least with a deterministic start state. This aligns with Fujimoto et al. (2022)'s arguments against residual minimization with finite data. This might also relate to SBEED's (Dai et al., 2017) and Trust-PCL's (Nachum et al., 2017) use of large batches.

We also find that RG works somewhat better on this offline Pendulum problem after adding an initial-value minimizing term like in Eq. 50 of Nachum & Dai (2020). We tried this value minimizer term with (semi-gradient) DQN too, and it did not fix DQN's divergence. That minimization term is different from CQL's minimization term, which minimizes the state-marginal of the dataset. CQL additionally includes a maximization term.

Moreover, we find RG works better here with a smaller value initialization or discount factor, though neither is a general solution.

B Where TRO does not help much

Gym Locomotion (Hopper, Walker, Half Cheetah). For TD3+BC on the gym locomotion tasks, we find TRO helps only somewhat. This might be due to a relative lack of return degradation for TD3+BC on these datasets, regardless of α. That is, on the locomotion datasets, we find excessive optimism (low α) does not tend to cause returns to peak and then drop. Rather, on the locomotion datasets, returns do not go up at all. That said, TRO can enable a more optimistic α setting and thus slightly higher returns on some environments, most prominently on `walker2d-random` where it almost doubles TD3+BC's score (not shown). Further, returns and the MSBE of the target net again anticorrelate, suggesting a route for improvement.

Brandfonbrener et al. (2021) show more frequent return degradation than we see on these tasks, though they use slightly different algorithms. For example, they use a single critic instead of clipped double Q-learning, and they use behavior cloning pre-training.

Adroit, Maze2D, and Kitchen. On these datasets (Fu et al., 2020; Rajeswaran et al., 2017; Gupta et al., 2019), we again do not find much benefit from TRO (not shown). Though, this might again be due to the limited return degradation for TD3+BC here, regardless of α. Further, we again see anticorrelation between returns and MSBE, which points toward the future work we mention in Section 6.

C Other Algorithmic Decisions

The three simplest binary algorithm design decisions we tested are below. Our main pseudocode, Algorithm 1, leaves out these design branches, showing only the algorithm we used for our main results. We name these Booleans such that all three are false in our main pseudocode.

- `immediate_target_update`: Within an algorithm step, update the target network immediately after the first of the two optimization steps, instead of delaying the target update until after both optimization steps.
- `critic_after_target`: On each algorithm step, run the critic gradient descent step after the TRO gradient descent step instead of before.

- `independent_batches`: On each algorithm step, sample two, independent batches, one for the TRO gradient descent step and one for the critic gradient descent step. This is as opposed to sampling one batch per algorithm step, using it for both gradient descent steps.

We show them in pseudocode in Algorithm 2. Typically, any combination of the decisions scored about equally well. We tested every combination on many environments, e.g., Seaquest, Antmaze, all of classic control, and `pen-human`. All results in this work delay the target update until after both optimization steps. All results in this work also run the critic step before the TRO step. Only the classic control tasks sample a new batch for the TRO step, but that decision had little to no effect in preliminary testing on many environments and datasets.

Algorithm 2 Other algorithmic decisions for Target Rate Optimization, again added to offline DQN. Changes from DQN are blue.

Given an offline dataset $\mathcal{D}$ of transitions $\{(s, a, r, s')\}$, and an initial soft target update rate, τ
Randomly initialize the parameters, θ, of an action-value function, Q
Initialize the target parameters $\bar{\theta} := \theta$
for each algorithm step **do**
  Update the target parameters $\bar{\theta} := \tau\theta + (1 - \tau)\bar{\theta}$
  Sample a batch $\{(s, a, r, s')\}$ from $\mathcal{D}$
  Compute the targets $\bar{y} := r + \max_{a'} Q_{\bar{\theta}}(s', a')$
  if not `critic_after_target` **then**
    Take one critic gradient descent step following $\nabla_\theta(\bar{y} - Q_\theta(s, a))^2$
    if `independent_batches` **then**
      Sample a second batch $\{(s, a, r, s')\}$ from $\mathcal{D}$
    end if
  end if
  if `immediate_target_update` **then**
    Update the target parameters $\bar{\theta} := \tau\theta + (1 - \tau)\bar{\theta}$
  end if
  Take one TRO gradient descent step following $\nabla_\tau \mathcal{L}(\tau)$, where
  $\mathcal{L}(\tau) := \begin{cases} (\bar{y} - Q_{\bar{\theta}}(s, a))^2 & \text{for TROT} \\ (\bar{y} - Q_\theta(s, a))^2 & \text{for TROP} \end{cases}$
  if `critic_after_target` **then**
    if `independent_batches` **then**
      Sample a second batch $\{(s, a, r, s')\}$ from $\mathcal{D}$
    end if
    Take one critic gradient descent step following $\nabla_\theta(\bar{y} - Q_\theta(s, a))^2$
  end if
end for

Other modifications that did not help in preliminary experiments include: a single, combined optimizer for the main net and τ; RAdam (Liu et al., 2019) for the critic or for TRO; and a larger ϵ for Adam.

D Alternative Handling for Double Q

We also briefly tested using two different τ optimizers, one for each critic separately. Using two τ optimizers prevented divergence at higher α values (more optimism) than a single τ optimizer allowed. This does not introduce any additional hyperparameter tuning, since both optimizers use the same TRO learning rate. The learned τs typically went in opposite directions. That is, typically, one τ increased while the other τ decreased. However, despite the added stability at higher α values, at no α value did using two optimizers score significantly higher than using one optimizer. We also tried using only a single critic (like DDPG), which similarly did not improve scores. We think it's possible that further work with two different τ optimizers might outscore a single τ optimizer.

E More Experiment Details

We leave states unmodified. That is, we do not normalize states, which the TD3+BC paper did for the gym locomotion tasks. Like that work, we found leaving states unmodified works better for Antmaze anyway. We did not bother testing Maze2D with normalized states.

Other than the pessimism, α, we leave all TD3+BC hyperparameters at their defaults, which is also the same as the TD3 defaults (Fujimoto & Gu, 2021).

We tuned TD3+BC's pessimism, α, in $\{1, 2, 2.5, 3, 4, 5, 10, 20, 30, 50, 100, 200, 500\}$, showing the best score per dataset. The original TD3+BC tuned over $\{1, 2, 2.5, 3, 4\}$, which does not suffice for the `antmaze-medium-*` and `antmaze-large-*` datasets. We discuss this sensitivity in Section 5.5. Our irregular hyperparameter search grid risks overfitting. Although such irregular searches are common in prior work, in future work we recommend, e.g., $\alpha \in \{2^{-1}, 2^0, \ldots, 2^9\}$ (Patterson et al., 2023).

We smooth all plots in all figures with rolling averages for readability. We average all scores over the final 10 evaluation steps.

Antmaze. On Antmaze, we used only a TRO learning rate in {1e-3, 1e-4}.

Following IQL (Kostrikov et al., 2021), CORL (Tarasov et al., 2022), and the original authors of the Antmaze dataset (Fu et al., 2020), we subtract one from the Antmaze rewards. Following CORL and the original TD3+BC paper, we normalize rewards on the gym locomotion tasks. We use the v2 versions of Antmaze. Some works report v0 scores, but the v0 Antmaze datasets have errors (Tarasov et al., 2023a).

Classic control. On the classic control tasks, we did not attempt to tune our TRO learning rate on any task except for Pendulum, where we show 1e-6. That is, we used only our default 3e-4 on CartPole and Acrobot.

For all algorithms, we searched over critic learning rates in {1e0, 1e-1, ..., 1e-6} and (initial) target update rates also in {1e0, 1e-1, ..., 1e-6}.

Atari. On Atari, we again did not attempt to tune our TRO learning rate at all.

We first ran 3M steps using critic learning rates in {1e-3, 1e-4, 1e-5, 1e-6}, and (initial) target update rates in {1e-3, 1e-4, ..., 1e-7}. We ran up to 10 seeds per configuration, concentrated on the best configurations. Then we ran 3 seeds for 50M steps for each of the best DQN configurations, which were critic learning rates in {1e-4, 1e-5, 1e-6}, and target update rates in {1e-5, 1e-6, 1e-7}. For TROT-DQN, we similarly ran 3 seeds for 50M steps for each of the best configurations, which were critic learning rates in {1e-4, 1e-5}, and initial target update rates in {1e-3, 1e-4, 1e-5, 1e-6}. Finally, we ran another 2 seeds for the best configuration for each algorithm. For DQN, that was a critic learning rate of 1e-5, and a target update rate of 1e-6. For TROT-DQN, that was a critic learning rate of 1e-5, and an initial target update rate of 1e-3, though TROT-DQN worked about equally well with all of those initial target update rates.