Analyzing and Overcoming Degradation in Warm-Start Reinforcement Learning

Benjamin Wexler, Elad Sarafian, and Sarit Kraus

***Abstract*— Reinforcement Learning (RL) for robotic applications can benefit from a warm-start where the agent is initialized with a pretrained behavioral policy. However, when transitioning to RL updates, degradation in performance can occur, which may compromise the robot's safety. This degradation, which constitutes an inability to properly utilize the pretrained policy, is attributed to extrapolation error in the value function, a result of high values being assigned to Out-Of-Distribution actions not present in the behavioral policy's data. We investigate why the magnitude of degradation varies across policies and why the policy fails to quickly return to behavioral performance. We present visual confirmation of our analysis and draw comparisons to the Offline RL setting which suffers from similar difficulties. We propose a novel method, Confidence Constrained Learning (CCL) for Warm-Start RL, that reduces degradation by balancing between the policy gradient and constrained learning according to a confidence measure of the Q-values. For the constrained learning component we propose a novel objective, Positive Q-value Distance (CCL-PQD). We investigate a variety of constraint-based methods that aim to overcome the degradation, and find they constitute solutions for a multi-objective optimization problem between maximimal performance and miniminal degradation. Our results demonstrate that hyperparameter tuning for CCL-PQD produces solutions on the Pareto Front of this multi-objective problem, allowing the user to balance between performance and tolerable compromises to the robot's safety.**

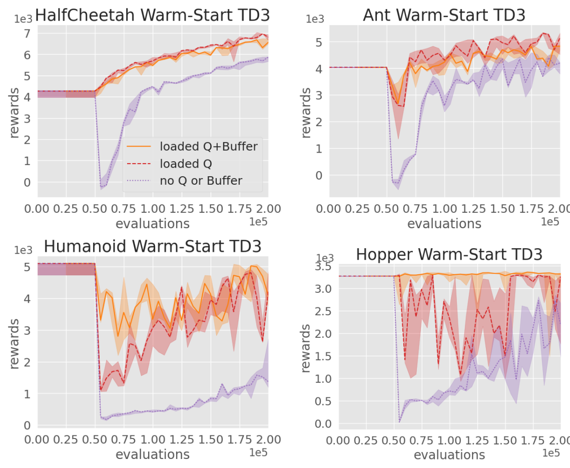

Fig. 1: WSRL initialized with a ∼4K reward expert over 4 seeds in 4 environments. *WSRL Degradation* in average episodic reward can be seen in the drop at 50K steps when the policy is first updated. Degradation is less severe when Q-net and/or buffer is loaded along with the actor (red, orange) versus when only the actor is loaded (blue). Similar results are found in the Fetch robotic environments.

I. Introduction

Reinforcement Learning (RL) has demonstrated success in simulated environments [1], [2], [3]. However, RL is challenging to implement in real-life robotic applications due to hard-to-define reward functions and the high rate of exploration required in limited time [4]. Unfortunately, robotic RL agents struggle to understand the transition dynamics of environments as intuitively as humans and have difficulty applying prior knowledge in an effective manner.

For example, consider the classic problem in robotics where a robot arm must pick up an object and place it in a goal location while receiving feedback from a sparse reward function (i.e., 1 if the object was picked up and placed at the goal and 0 otherwise). Modern RL algorithms fall short in this seemingly simple task because large amounts of interaction are required to acquire a positive feedback signal [5]. Furthermore, without a sophisticated reward shaping function the probability of stumbling upon the goal through naive exploration alone may be extremely small. Additionally, in real-world environments, where interactions are costly, such a computationally-intensive algorithm becomes unrealistic. Excessive exploration using actions with uncertain outcomes can impact safety within the environment and inflict wear and tear on the robot as well.

To overcome these issues, many approaches suggest leveraging expert data. These involve using expert demonstrations in RL: to guide and regularize exploration [6]; as ground-truth to recover a reward function [7]; as examples in the replay buffer [8]; or as targets for a Behavior Cloning loss. In the latter, this term is combined with the RL objective [9], or with the RL objective and L2 Regularization [10].

Another approach is Warm-Start RL (WSRL) [11], [12] where learning consists of initializing the agent with a pretrained behavioral policy and improving upon its already achieved moderate performance through RL. These initial policies can be created using Imitation Learning on expert demonstrations or by obtaining a previouly trained RL policy. This approach to incorporating expert knowledge into RL can overcome the problem of sparse rewards, increase the sample efficiency, reduce the training time, and improve the safety of RL policies. Initializing RL with a pretrained behavioral policy should also ensure better and more robust performance of the robotic agent and allow for safe widespread deployment of self-learning robots in varied and dynamic environments.

Benjamin Wexler, Elad Sarafian and Sarit Kraus are with the Department of Computer Science at Bar-Ilan University, Ramat-Gan, Israel `benwex93@gmail.com` `elad.sarafian@gmail.com` `sarit@cs.biu.ac.il`. This research has been partly supported by the EU Project TAILOR under grant 952215 and the DSI at BIU.

In our problem formulation, we assume no previous knowledge of the value function is available. the value function must be approximated using a neural network by training it from scratch. In practice, however, we find that initializing **off-policy** algorithms like DDPG [13], SAC [14], or TD3 [15] with a policy that achieves moderate-to-expert performance results in an initial severe degradation of the policy. We refer to this phenomenon as *WSRL Degradation*. This problem is intrinsically linked to known complications associated with Offline RL, first described in [16]. Our proposed method to overcome *WSRL Degradation* is inspired by some of the recent Offline RL algorithms: Batch-Constrained Q-Learning (BCQ) [17], BEAR [18], Behavior-Regularized [19][20][21], Rerouted-Behavior-Improvement [22], Policy-Constraint [23], KL-Control [24], and Critic Regularization [25]. Like our solution, these algorithms share the core idea that the learned policy should be constrained to the behavioral policy. Our method uses a constraint that modifies the learning gradient between IL and RL, a technique which bears similarity to PCGrad proposed in [26].

The contributions of our paper are as follows:

1) We demonstrate that *WSRL Degradation* is a common phenomenon by running WSRL across several environments and experts that attain various levels of reward.
2) We analyze why degradation occurs, where it is more severe, and pinpoint the specific causes.
3) We implement several constraint methods, including Offline RL and Online RL solutions and find that all methods introduce a trade-off between degradation and final performance.
4) We propose CCL-PQD which constrains updates based on a confidence measure of the value function. We demonstrate that hyperparameter selection results in solutions that lie on the Pareto Front that balances maximimal performance and mininimal initial drop and solves this multi-objective problem [27].

II. Analyzing *WSRL Degradation*

We demonstrate *WSRL Degradation* in four continuous control environments: Ant-v3, HalfCheetah-v3, Hopper-v3, and Humanoid-v3 [28]. We begin by training a TD3 agent and halting once the agent achieves a set level of performance but with room for improvement. We then copy the policy into a new TD3 agent without the value function, roll out 50K steps to initially train the Q-net, and then begin updating the policy. We observe the degradation after the policy updates begin [fig. 1]. This process of isolating the RL policy by discarding the value function and replay buffer simulates the scenario where there is access to an expert policy through IL. We use TD3 as the off-policy RL algorithm for our experiments but other methods built on the same fundamental core, like SAC and DDPG, exhibit similar degradation.

While the policy ultimately recovers, this degradation is still undesirable because ideally the robotic agent would maintain its performance to avoid damage to itself and the surrounding environment. We note that for some policies, there is no large dip in performance. For example, runs initialized with $\sim$2K discounted reward experts in the HalfCheetah environment often experience lower degradation. However, for the majority of runs in most environments there is severe *WSRL Degradation*.

We also observe that it takes many iterations ($\sim$500K) before the policy recovers to the original behavioral performance. However, WSRL does converge faster than vanilla off-policy RL (1M Iterations). Finally, we do not observe the same degradation in Warm-Start **on-policy** RL using methods such as VPG [29], PPO [30], and PPG [31].

To better understand *WSRL Degradation* we need to address three key questions:

1) What causes the initial degradation?
2) Why is the degradation smaller for some lower reward behavioral policies?
3) Why does it take so long for the agent to recover to the initial behavioral performance?

A. Comparing WSRL to Offline RL

Results from WSRL where the Q-net is loaded compared to vanilla WSRL[fig. 1] indicate that the degradation of the policy is related to the initial training of the Q-net. Training the Q-net on an essentially static dataset (because the policy is not updated) is reminiscent of the **Offline** RL setting; we compare the two and identify common issues and solutions. In Offline RL, an agent is trained using a buffer of offline trajectories (produced by a behavioral agent) while being unable to interact with the environment. Rather than simply cloning the actions with highest expected discounted reward, the agent is expected to optimize the policy by maximizing a value function trained on this offline dataset. It is observed that **off-policy** RL algorithms fail when deployed on static datasets without online interaction. Two key papers, BCQ [16] and BEAR [18], provide explanations for the inability of off-policy RL to learn on a static dataset offline and propose solutions to overcoming the problem.

B. Extrapolation Error

Both BCQ and BEAR identify Extrapolation Error (EE) as the source of off-policy RL's inability to learn offline [18] [17]. EE is described as the error induced by the mismatch between state-action pairs in the behavioral dataset and state-action pairs that have no reward data. We refer to the latter as Out-of-Distribution (OOD) data. The value function $Q(s, a)$ where action a was never taken should be undefined. However, because there is no notion of undefined values in value functions modelled as a neural network approximators, $Q(s, a)$ outputs defined values even though there are no data for such state-action pairs. These errant values negatively impact the off-policy RL algorithm, biasing the model when calculating the Bellman-Backup $|Q - \mathcal{T}Q|$ and the policy gradient $\nabla_a Q(s, a)$. We attribute the error to OOD actions rather than OOD states, because all network updates use fixed states s_β (i.e., only states seen in the behavioral policy data).

C. Analyzing Extrapolation Error in Warm-Start RL

In WSRL, the behavioral policy is pre-loaded but the Q-net (denoted Q_{NL}) is not loaded and is instead initialized

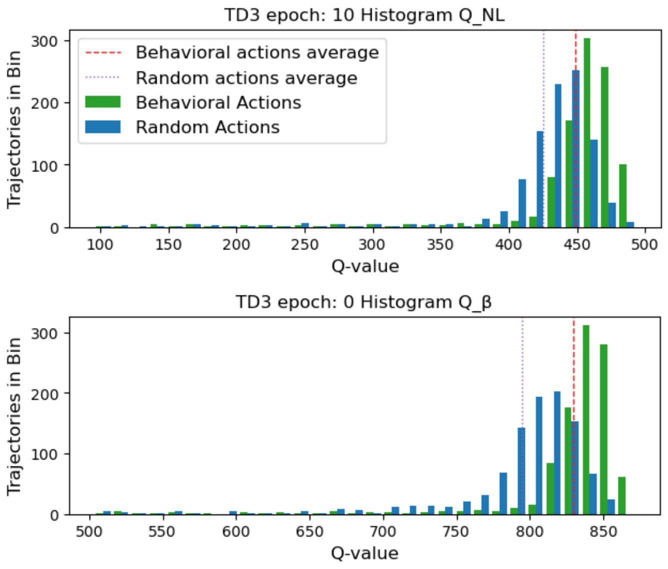

Fig. 2: **Top:** Output of Q_{NL} on behavioral actions (green) vs. random actions (blue) over one sampled episode. Both classes of actions tend to have large Q-values under the same state distribution despite the agent never taking these random actions. **Bottom:** Output of Q_β on behavioral actions (green) vs. random actions (blue) over one sampled episode. Similar to the output of Q_{NL}, both classes of actions tend to have large Q-values under the same state distribution.

with random parameters and then pre-trained offline on behavioral data during the initial phase. We analyze[1] Q_{NL}'s performance on a static dataset by comparing its output on **behavioral** actions a_β against its output on **random** actions a_r under the state distribution induced by the behavioral policy π_β over one sampled episode.

We can see that random actions generally have unusually high Q-values [fig. 2, top] suggesting that the Q-net extrapolates high values to most actions, even if they have never been taken. This generalization can be attributed to state s_β being consistent across inputs $Q_{NL}(s_\beta, a_\beta)$ and $Q_{NL}(s_\beta, a_r)$. This generalization can cause problems during training and is the source of EE. [32] explain that EE is a common phenomenon that affects Neural Networks citing [33] who describe how an insufficient data distribution can drive over-generalization.

As a baseline, we visualize [fig. 2, bottom] the properly trained original behavioral Q_β's performance on a static dataset by comparing its output on behavioral actions a_β against its output on random actions a_r under the same state distribution induced by the behavioral policy π_β over one sampled episode.

To visualize EE in Q_{NL} we sample a fixed behavioral state s_β with action a_β, and plot the output $Q_{NL}(s_\beta, a_\beta+\epsilon)$ where $\epsilon \sim U(0,1)$. Actions are reduced from a six-dimensional vector to two dimensions through Principal Component

[1]Our analysis of WSRL uses a HalfCheetah expert of ∼10K cumulative episodic reward where *WSRL Degradation* is most severe; however, results are consistent for ∼4K HalfCheetah reward experts as well as other environments.

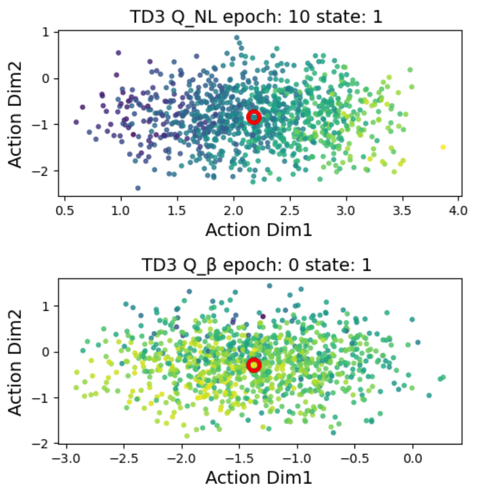

Fig. 3: **Top:** $Q_{NL}(s_\beta, a_\beta)$ (circled, center) vs. $Q_{NL}(s_\beta, a_\beta+\epsilon)$, where the behavioral policy action has lower Q-value than some OOD actions (shown as lighter-colored points on the right). **Bottom:** $Q_\beta(s_\beta, a_\beta)$ (circled, center) vs. $Q_\beta(s_\beta, a_\beta+\epsilon)$ where the behavioral policy action still has slightly lower Q-value than some OOD actions (to the left).

Analysis (PCA). These results [fig. 3, top] demonstrate EE where Q_{NL} assigns higher values for many OOD actions in a given state than for the behavioral action. Ultimately, the policy will be updated in the direction of those OOD actions with higher Q-values.

As a baseline, we fix the behavioral state and plot the output $Q_\beta(s_\beta, a_\beta+\epsilon)$ [fig. 3, bottom]. We first remark that the output range is lower for Q_{NL} and rises as training continues but it is the relative values that are the most important. We then note the presence of high Q-values in OOD actions for both Q_{NL} and Q_β. This similarity suggests that EE should also be present in the loaded behavioral value function Q_β. Why then does training with Q_β lead to stable policy updates whereas Q_{NL} results in severe degeneration in policy performance?

We analyze the relative values of $Q_{NL}(s_\beta, a_\beta)$ and $Q_\beta(s_\beta, a_\beta)$ compared to $Q_{NL}(s_\beta, a_\beta+\epsilon)$ and $Q_\beta(s_\beta, a_\beta+\epsilon)$. We find that $Q_{NL}(s_\beta, a_\beta)$ usually plots near the center of the $Q_{NL}(s_\beta, a_\beta+\epsilon)$ distribution [fig. 4, top] (note that the PCA graphs in fig. 3 corresponds to the left-most histograms in fig. 4). In contrast, $Q_\beta(s_\beta, a_\beta)$ is generally higher than most $Q_\beta(s_\beta, a_\beta+\epsilon)$ [fig. 4, bottom]. This difference is also manifest when loading a behavioral policy of ∼4K cumulative reward where $Q_{NL}(s_\beta, a_\beta)$ is in the 46th percentile on average versus $Q_\beta(s_\beta, a_\beta)$ in the 66th percentile on average.

That $Q_{NL}(s_\beta, a_\beta)$ plots in the center of the distribution indicates that the value-function is not well trained on OOD actions and assigns them similar values that are normally distributed. In contrast, Q_β clusters the values of most OOD actions below that of a_β reflecting higher knowledge of the surrounding action space likely due to Q_β having been trained on a wide range of actions in the same state distribution, as well as on actions in sub-optimal states. Q_β's

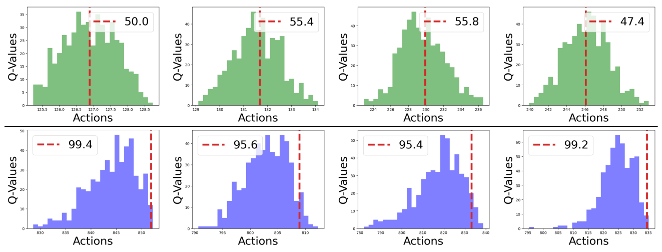
Fig. 4: **Top:** Histogram (top left) corresponding to PCA graph [fig. 3, bottom] with histograms of 3 other randomly sampled states. $Q_{NL}(s_\beta, a_\beta)$ is in the 52nd percentile on average and is not maximal compared to $Q_{NL}(s_\beta, a_\beta + \epsilon)$. **Bottom:** Histogram (bottom left) corresponding to PCA graph [fig. 3, top] with histograms of 3 other randomly sampled states. $Q_\beta(s_\beta, a_\beta)$ are in the 95th percentile on average and are maximal compared to $Q_\beta(s_\beta, a_\beta + \epsilon)$.

wider range of knowledge mitigates EE and results in a policy gradient $\nabla_\theta Q_\beta(s, \pi_\theta(s))$ that is more robust.

We can now answer our first question as to why the policy initially degrades. Due to OOD generalization, the Q-net returns higher values for some actions never taken by the partially trained behavioral policy, i.e., $Q(s_\beta, a_\beta + \epsilon_{action}) > Q(s_\beta, a_\beta)$ for those actions. The policy gradient $\nabla_\theta Q_\phi(s, \pi_\theta(s))$ will therefore point towards these OOD actions and the policy will be updated to take unknown actions and degrade.

D. Gradient Error due to Extrapolation Error

To answer the second question as to why the degradation is smaller for some lower reward behavioral policies we compare the norm of the gradients $||\nabla_a Q(s,a)||$ of WSRL for ∼4K versus ∼10K cumulative reward in the HalfCheetah environment. When training the Q-net offline we observe that the average norm of the gradient for the higher reward expert is larger (∼0.08) than that of the lower reward expert (∼0.04). As $Q(s,a)$ is trained on higher reward trajectories, the gradients $\nabla Q(s,a)$ will be larger. Formally, for $c > 1$, if

$$R_{10K} = \sum_{i=t}^{T} \gamma^{i-t} r_{10K} = c \cdot R_{4K} > R_{4K} = \sum_{i=t}^{T} \gamma^{i-t} r_{4K} \tag{1}$$

and the Q-function is trained to output higher values for $(s,a) \sim \tau$

$$\begin{aligned} Q^{\pi_{10K}}(s,a) = \underset{\tau \sim \pi_{10K}}{E}[R_{10K}(\tau)] = c \cdot Q^{\pi_{4K}}(s,a) > \\ Q^{\pi_{4K}}(s,a) = \underset{\tau \sim \pi_{4K}}{E}[R_{4K}(\tau)] \end{aligned} \tag{2}$$

and the inequality holds under differentiation and under the norm which is also positive:

$$||\nabla Q^{\pi_{10K}}(s,a)|| = c \cdot ||\nabla Q^{\pi_{4K}}(s,a)|| > ||\nabla Q^{\pi_{4K}}(s,a)|| \tag{3}$$

This explains why the degradation is less severe for the lower reward expert since the lower gradient is itself a constraint on the agent to not take large update steps.

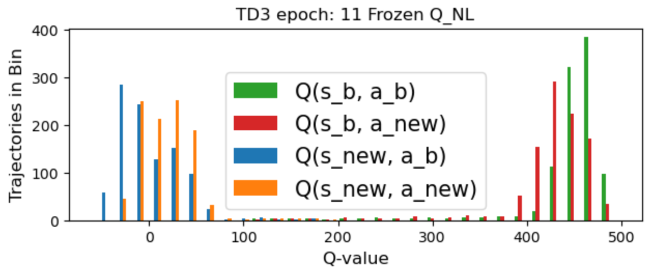

Fig. 5: Histogram of the output of frozen Q_{NL}.

We can scale down the policy gradient to some threshold by the norm s.t. for $g = \nabla_\theta Q_\phi(s, \pi_\theta(s))$: if $||g|| > threshold$: then $g \leftarrow \frac{threshold \cdot g}{||g||}$. This ensures that the norm of the updated gradient will equal the threshold since $\left|\left|\frac{threshold \cdot g}{||g||}\right|\right| = \frac{threshold}{||g||} \cdot ||g|| = threshold$.

This results in reduced degradation, though the improvement in performance over time is proportional to the threshold of the gradient clipping. Scaling the gradient by lowering the learning rate η s.t. $\theta_{t+1} = \theta_t + \eta \cdot \nabla_{\theta_t} Q_\phi(s, \pi_{\theta_t}(s))$ also results in less degradation. These methods are effective at overcoming degradation, but come with the trade-off of slower learning and require tuning through trial-and-error.

To answer the third question as to why it takes so long for the agent to recover to its original performance, we freeze updates on Q_{NL} after the Offline training phase and update the policy from π_β to an updated version π_U by training for one epoch. We examine the output of Q_{NL} over a sampled episode of: behavioral states and actions $(s_\beta, \pi_\beta(s_\beta))$; behavioral states and updated policy actions $(s_\beta, \pi_U(s_\beta))$; updated states along with actions sampled from a separate behavioral trajectory $(s_U, \pi_\beta(s_\beta))$; and updated policy states and actions $(s_U, \pi_U(s_U))$.

We first observe that only behavioral state input to the Q-net results in EE [fig. 5, right] in contrast to behavioral action input which is assigned significantly lower value [fig. 5, left]. We then note the large magnitude of the update by considering that the same Q-net used for updating the policy gave lower evaluation for policies after being updated, i.e., $Q(s_\beta, a_\beta) > Q(s_\beta, \pi_{U=\beta+\eta \cdot \nabla_{a_\beta} Q(s_\beta, a_\beta)}(s_\beta))$. This indicates that the update is so large that it shifts the policy OOD, as confirmed by the Q-net assigning these actions larger values than if the policy had remained in the behavioral state distribution. Instead, the values assigned are closer to the default random values in the range $[-1, 1]$ used to initialize the Q-net. Once the policy has radically deviated from the behavioral state distribution to one where it has no prior knowledge, the policy gradient will no longer be a reliable indicator. The agent will have no data regarding actions that can return it to the behavioral distribution and the probability of re-encountering a behavioral state is low due to the high dimensionality of the state space. To recover fully, the policy will have to explore and learn Q-values for the new state distribution in order to find a reliable gradient towards a better policy. Some gradient signal from behavioral policy data is retained however, because performance converges faster than training from scratch.

E. Bootstrapping Error due to Extrapolation Error

The inability to recover behavioral performance is also related in part to why off-policy algorithms fail to learn offline. We follow [23] who explain that the issue lies with the calculation of the Bellman Backup $|Q - \mathcal{T}Q|$. When training the Q-net offline, the update step utilizes an untrained policy's predicted action $\pi(s'_\beta)$ to bootstrap onto $Q(s'_\beta, \pi(s'_\beta))$ that is never directly updated via the Bellman Backup since π is a randomly initialized policy that returns an action not in the behavioral distribution. As demonstrated in our work, $Q(s'_\beta, \pi(s'_\beta))$ may be assigned a value close to $Q(s'_\beta, a'_\beta))$ where a'_β is the action actually taken in the next state. However, maximizing over an average Q-value for s'_β without discriminating between particular actions will result in a policy gradient $\nabla_a Q(s'_\beta, a = \pi(s'_\beta))$ that updates the policy to take erroneous actions that can only be corrected through online trial and reward feedback, a option unavailable in off-policy RL.

WSRL also contains an offline phase where the Q-net is trained on behavioral trajectories (with added exploration noise and without access to reward data) before the policy is updated. However, Offline RL is initialized with a random policy whereas WSRL is initialized with a behavioral policy where $\pi(s'_\beta) = a'_\beta$ and therefore the value $Q(s'_\beta, \pi_\beta(s'_\beta))$ can properly propagate through the Bellman Backup. Where Bootstrapping Error comes into play is in the online phase once the state distribution has already shifted such that s' is no longer in the behavioral distribution resulting in a bootstrapped target $Q(s', \pi(s'))$ that has not yet been updated through reward feedback. This generates a poor Q-net estimate and therefore an unreliable policy gradient that prevents the policy from quickly returning to behavioral performance. Learning the true values of OOD trajectories is the only way of returning to better performance, but this is constrained by the low joint probability of selecting an OOD trajectory from the buffer given that each prior trajectory has been sampled and propagated via Bellman Backup.

III. Overcoming *WSRL Degradation*

After analyzing it's source, we now aim to develop methods that reduce *WSRL Degradation* and as a result guarantee the safety and reliability of the robotic agent. To achieve this we limit the probability of the policy entering a shifted state distribution which reduces the effect of EE by preventing the policy from becoming stranded in a region where it has no prior knowledge and cannot return to the behavioral distribution. We build upon solutions proposed for Offline RL, which typically constrain agent updates to keep the policy close to the behavioral policy. This can be achieved in two ways: directly constraining the policy or introducing a penalty during updates.

A. Offline RL Constraint Methods

One implementation of policy constraint is an agent that learns a layer ξ_θ that perturbs behavioral actions and is constrained within some bounds $[-\alpha, \alpha]$. This is the approach in BCQ [16] where the policy update is calculated as:

$$\max_\theta E\left[Q_\phi(s, a_\beta + \xi_\theta(s, a_\beta))\right] \tag{4}$$

Another approach is to constrain the policy updates:

$$\max_\theta E\left[Q_\phi(s, \pi_\theta(s))\right] s.t.[D(a_\beta, \pi_\theta(s))] \leq \epsilon \tag{5}$$

where D is a distribution distance metric. This method is used in BEAR [18] where D is *Kernel-MMD* [34].

B. Offline RL Penalty Methods

Another approach is to constrain updates by introducing a penalty on values that lie further from the behavioral distribution. This penalty, represented as a distribution distance D between the agent and behavioral policy, can be introduced in the Q-net update by minimizing ϕ over:

$$\mathrm{E}\left[\left|r + \gamma\left(Q_{\phi'}(s', a'_\beta) - \alpha D\left(a'_\beta, \pi_\theta(s')\right)\right) - Q_\phi(s, a_\beta)\right|^2\right]$$

or in the policy update objective:

$$\max_\theta E\left[Q_\phi(s, \pi_\theta(s)) - \alpha D(a_\beta, \pi_\theta(s))\right] \tag{6}$$

This method is used in [24] and [19] which define D as either KL-divergence, *Kernel-MMD*, or Wasserstein Distance.

C. Online RL Constraint Methods

As a baseline, we implement **online** versions of Offline RL constraint and penalty methods across a range of hyperparameter values α. These include a BC penalty on the policy update; with a learned perturbation policy (analogous to online BCQ); and with a BC penalty on the Q-net update. We found degradation was significantly reduced in the first two methods but not in the latter for any $\alpha \in [3e\text{-}9, 3e\text{-}8, ..., 3e\text{-}1]$. Lowering the learning rate or reducing the gradient are other effective online constraint methods (we omit other offline RL methods such as BEAR and CQL since they do not translate readily to the online setting as we assume access only to the expert policy and not the rewards). All of these methods, however, may still be subject to degradation, require arbitrary parameters for tuning, and result in slower learning that is inversely related to the constraint. In addition, WSRL differs from Offline RL in that the policy can interact with the environment and improve over the behavioral policy. Rather than simply implementing an online version of an Offline RL algorithm that maintains a constant constraint, we prefer a method that slowly relaxes the constraint over time eventually reducing to vanilla off-policy RL.

IV. Confidence Constrained Learning for Warm-Start off-policy RL

We propose Confidence Constrained Learning (CCL) for WSRL which uses a constrained Offline RL algorithm as a basis, and a scheduler that relaxes the constraint over time according to an appropriate metric, ultimately transitioning to vanilla off-policy RL. For our base constrained algorithm we selected a form of Policy Penalty defined as:

$$\alpha Q(s, \pi(s)) + (1 - \alpha) Distance(a_\beta, a) \tag{7}$$

TABLE I: Comparing CCL-PQD to Warm-Start and vanilla TD3 in HalfCheetah with regards to safety and performance.

Env	Method Rule	Expert Reward	α-*start*	Min	Max	Avg	VaR (95)	VaR (99)	VaR (99.9)	CVaR (95)	CVaR (99)	CVaR (99.9)
Half	TD3	0		-576.9	**11171.0**	**7478.7**	-4.3	-55.4	-156.0	-44.3	-155.5	-779.9
Cheetah	WS-TD3	4K		-589.2	9625.8	7180.2	-5.6	-49.7	-1611.2	-189.1	-886.0	-8056.2
	CCL-PQD	4K	1e-2	**2689.8**	8520.6	6193.6	**-4.2**	**-7.4**	**-8.6**	**-6.6**	**-11.9**	**-42.8**
Half	TD3	0		-576.9	11171.0	7478.7	-4.3	-55.4	-156.0	-44.3	-155.5	-779.9
Cheetah	WS-TD3	10K		-1084.4	**11381.3**	8864.9	-12.0	-41.9	-1775.1	-206.8	-959.3	-8875.5
	CCL-PQD	10K	1e-4	**7393.4**	10875.4	**10168.6**	**-3.5**	**-4.7**	**-5.3**	**-4.7**	**-7.6**	**-26.7**

where α is the constraint factor that facilitates interpolation between the vanilla policy gradient ($\alpha = 1$), where degradation compromises the robot's safety, and imitating the behavioral policy ($\alpha = 0$), where safety is guaranteed but performance does not increase.

The metric used by the scheduler to update α is a confidence measure of the Q-net corresponding to the EE that allows for large policy updates when the risk of EE is low and constrains the update otherwise. This confidence measure P is the percentile of the Q-net output on sampled trajectories $Q(s, a)$ compared to $Q(s, a + \epsilon)$ defined as:

$$P = \frac{N(Q(s,a) > Q(s, a+\epsilon))}{N(Q(s, a+\epsilon))} \tag{8}$$

We map the percentile, from the values between completely random and full certainty $[0.5, 1]$, to the range of α values between the minimal safe policy penalty and unconstrained vanilla policy gradient $[\alpha\_start, 1]$. We found that a logarithmic scale is preferable for the mapping where α ranges from $10^{[\log_{10}(\alpha\_start), 0]}$.

A. Positive Q-value Distance

There are many candidates when choosing a distance function; we define the Positive Q-value Distance (PQD) which modifies an auxiliary distance function $D(a_\beta, a)$. The standard update vector for the distance is:

$$\mathbf{v} = \text{-}\nabla_a D(a_\beta, a) \cdot \nabla_\phi \pi_\phi(a|s) \tag{9}$$

For better results we want to reduce the distance to actions with the highest Q-return and ignore poor actions produced by exploration. In particular, we utilize the components of the distance function gradient that do not decrease the average Q-value of the actions and therefore our update will be $\mathbf{v} = \mathbf{q}^{PQD} \cdot \nabla_\phi \pi_\phi(a|s)$ where

$$\mathbf{q}^{PQD} = \begin{cases} \nabla_a D(a_\beta, a) \perp \text{-}\nabla_a Q^\pi_\theta(s,a) & \nabla_a D \cdot \text{-}\nabla_a Q \le 0 \\ \nabla_a D(a_\beta, a) & \nabla_a D \cdot \text{-}\nabla_a Q > 0 \end{cases} \tag{10}$$

where $\nabla_a D \cdot \text{-}\nabla_a Q$ is a shorthand for $\nabla_a D(a_\beta, a) \cdot \text{-}\nabla_a Q^\pi_\theta(s,a)$ and $\nabla_a D(a_\beta, a) \perp \text{-}\nabla_a Q^\pi_\theta(s,a) =$

$$\nabla_a D(a_\beta, a) - \frac{\text{-}\nabla_a Q^\pi_\theta(s,a)}{||\text{-}\nabla_a Q^\pi_\theta(s,a)||^2} \nabla_a D(a_\beta, a) \cdot \text{-}\nabla_a Q^\pi_\theta(s,a) \tag{11}$$

Intuitively, where $\nabla_a D \cdot \text{-}\nabla_a Q > 0$, the distance gradient overlaps with the policy gradient in the same direction for those components of $proj_{\text{-}\nabla_a Q} \nabla_a D$. This follows from the scalar projection where $(\vec{a} \cdot \vec{b})/||\vec{b}||^2 > 0 \iff \vec{a} \cdot \vec{b} > 0$. This implies that following the distance gradient for those components will not decrease the Q-value average of the policy. However, following distance gradient components s.t. $\nabla_a D \cdot \text{-}\nabla_a Q \le 0$, will decrease the Q-value average of the policy. Therefore, we follow the vector rejection $\nabla_a D$ onto $\text{-}\nabla_a Q$ defined as $\vec{a} - (\vec{a} \cdot \vec{b})/||\vec{b}||^2 \cdot \vec{b}$ which restricts the components of the distance gradient to those that decrease the distance but do not decrease the Q-value average.

When integrating PQD into CCL we modify our original objective such that the final update vector is:

$$(1-\alpha)\mathbf{q}^{PQD} + \alpha \mathbf{q}^{\overline{PQD}} \cdot \nabla_\phi \pi_\phi(a|s) \tag{12}$$

where $\mathbf{q}^{\overline{PQD}} = \nabla_a Q(s,a) - \mathbf{q}^{PQD}$, the remaining policy gradient.

V. CCL-PQD Evaluation

We evaluate CCL with PQD as the primary distance function (CCL-PQD) and BC as the auxiliary distance function, and assess CCL-PQD performance with respect to safety and performance. For our experiments we experiments a budget of 1M timesteps which may not result in convergence for all environments but more accurately reflects real-life scenarios. We compare results against vanilla TD3 and its Warm-Start counterpart in [fig. 6] and [Table I]. A higher Min value indicates less degradation and a safer robotic agent, and a higher Max value indicates better overall performance. We also evaluate Value at Risk (VaR) and Conditional Value at Risk (CVaR) as additional safety metrics as proposed in [35]. We find CCL-PQD to be a powerful method for maintaining agent safety while increasing performance since it obtains the best results for VaR, CVaR, and highest minimal drop (see bolded values) while remaining competitive with the other methods in terms of average and maximum reward.

While CCL-PQD does not always achieve the highest maximum and average reward, it still eliminates or significantly reduces degradation. In constraint methods, it is common to expect a trade-off between degradation and learning over time. Selecting a constraint method and tuning its constraint factor translates to solving a multi-objective optimization problem where the set of best solutions lie on the Pareto Front that best balances maximimal performance and mininimal initial drop [fig. 7]. Our results, from many robotic environments, indicate that CCL-PQD often spans the Pareto Front across the possible values for the constraint factor, ensuring that the permitted risk to the robotic agent's

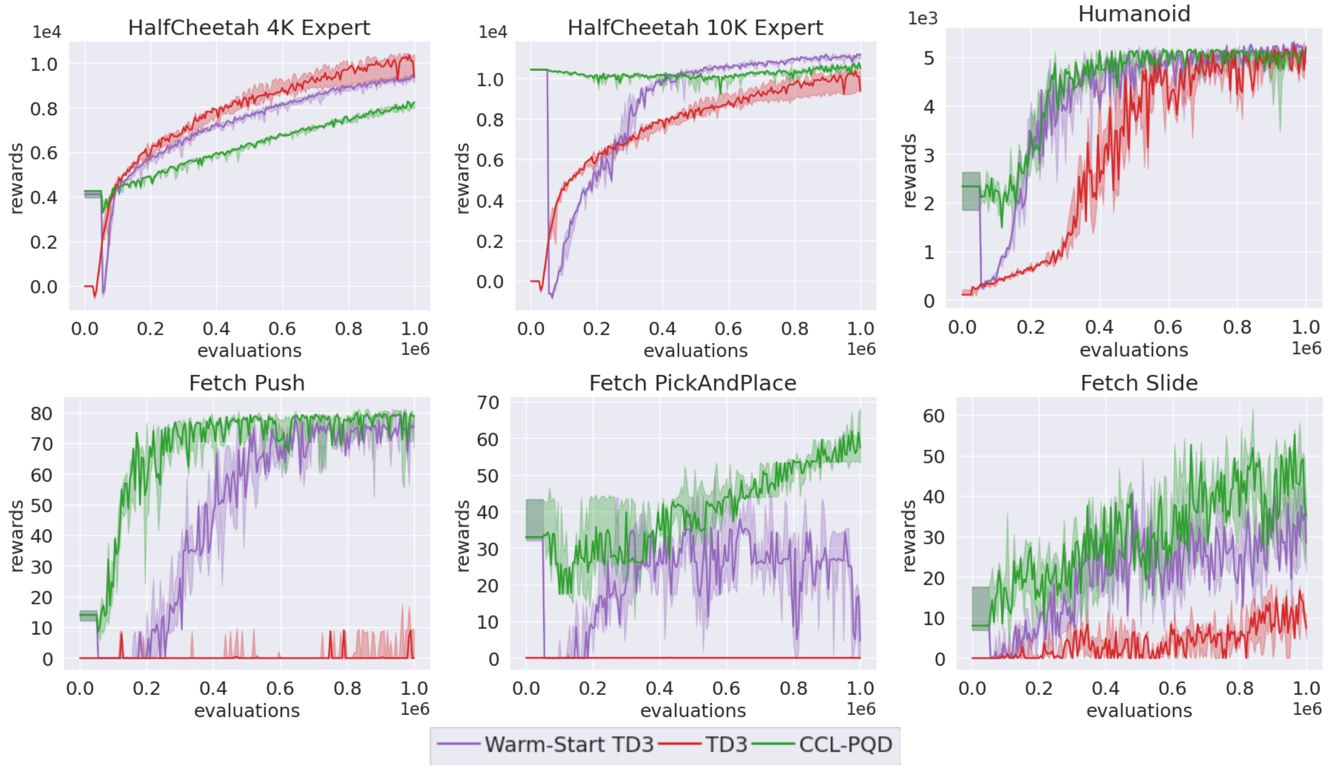

Fig. 6: CCL-PQD compared to Warm-Start and vanilla TD3. We generate ϵ over 500 samples. CCL-PQD minimizes degradation and maintains safety of the robotic agent while increasing performance.

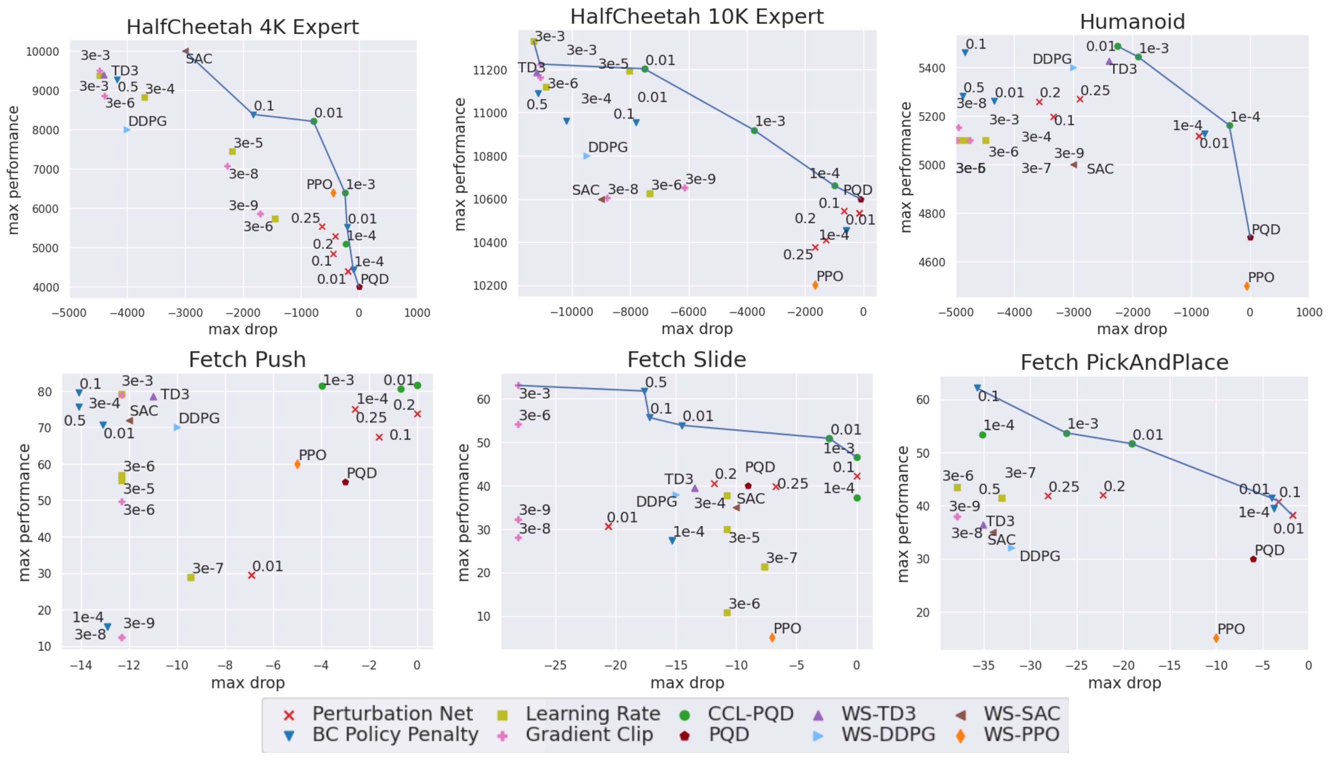

Fig. 7: Comparing constraint methods with various constraint factors α. CCL-PQD often spans the Pareto Front.

safety in favor of performance gains is optimal. Other constraint methods usually plot behind the Pareto Front with Warm-Start on-policy RL (PPO) producing less degradation but lower final performance and Warm-Start off-policy RL (DDPG, TD3, SAC) resulting in more degradation but higher performance in the long run. We note that BC Policy Penalty appears at times on the Pareto Front but not as consistently as CCL-PQD. We also observe in our experiments in the Fetch Push environment that solutions do not produce a visible Pareto Front since CCL-PQD is optimal for both objectives. The results also include an ablation test of PQD that demonstrate its effectiveness as a safe optimization objective with minimal degradation but slow incremental improvement. Finally, our results demonstrate that CCL-PQD is an effective method for improving upon behavioral policies in a safe manner without degradation, not only in dense reward environments such as HalfCheetah and Humanoid, but in sparse reward robotic environments like Fetch as well.

VI. Conclusion

The goal of our work was to study *Warm-Start RL Degradation*, analyze its root cause, and develop a solution to overcome it in robotic environments. We ascribed this degradation to Extrapolation Error, demonstrated why degradation is worse for higher reward environments due to Gradient Error, and explained why the policy fails to return quickly to behavioral performance due to Bootstrapping Error and distributional shift. We proposed CCL combined with a novel metric PQD that does not always achieve the highest maximum and average reward but eliminates or significantly reduces degradation, a critical component for real-life robotics. We demonstrated the trade-off that exists for many constraint methods where higher performance is possible when more degradation is risked. CCL-PQD often performs along the Pareto Front giving the user the power to optimize this trade-off in a wide range of applications.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv:1912.06680*, 2019.

[4] H. Zhu, J. Yu, A. Gupta, D. Shah, K. Hartikainen, A. Singh, V. Kumar, and S. Levine, “The ingredients of real-world robotic reinforcement learning,” *arXiv:2004.12570*, 2020.

[5] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in *NeurIPS*, 2017, pp. 5048–5058.

[6] S. Christen, S. Stevsic, and O. Hilliges, “Guided deep reinforcement learning of control policies for dexterous human-robot interaction,” in *ICRA*. IEEE, 2019, pp. 2161–2167.

[7] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *ICML*, 2004, p. 1.

[8] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” *arXiv:1707.08817*, 2017.

[9] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” in *ICRA*. IEEE, 2018, pp. 6292–6299.

[10] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, “Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments,” *arXiv:1910.04281*, 2019.

[11] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, “Fast policy learning through imitation and reinforcement,” *arXiv:1805.10413*, 2018.

[12] F. Zhu and P. Liao, “Effective warm start for the online actor-critic reinforcement learning based mhealth intervention,” *arXiv:1704.04866*, 2017.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv:1509.02971*, 2015.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” *arXiv:1801.01290*, 2018.

[15] S. Fujimoto, H. Van Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” *arXiv:1802.09477*, 2018.

[16] S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau, “Benchmarking batch deep reinforcement learning algorithms,” *arXiv:1910.01708*, 2019.

[17] S. Fujimoto, D. Meger, and D. Precup, “Off-policy deep reinforcement learning without exploration,” in *ICML*, 2019, pp. 2052–2062.

[18] A. Kumar, J. Fu, G. Tucker, and S. Levine, “Stabilizing off-policy q-learning via bootstrapping error reduction.” [Online]. Available: http://arxiv.org/abs/1906.00949

[19] Y. Wu, G. Tucker, and O. Nachum, “Behavior regularized offline reinforcement learning,” *arXiv:1911.11361*, 2019.

[20] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans, “Algaedice: Policy gradient from arbitrary experience,” *arXiv preprint arXiv:1912.02074*, 2019.

[21] A. Kumar, A. Zhou, G. Tucker, and S. Levine, “Conservative q-learning for offline reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[22] E. Sarafian, A. Tamar, and S. Kraus, “Constrained policy improvement for efficient reinforcement learning.” in *IJCAI*, 2020, pp. 2863–2871.

[23] S. Levine, A. Kumar, G. Tucker, and J. Fu, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” *arXiv:2005.01643*, 2020.

[24] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard, “Way off-policy batch deep reinforcement learning of implicit human preferences in dialog,” *arXiv:1907.00456*, 2019.

[25] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum, “Offline reinforcement learning with fisher divergence critic regularization,” in *ICML*, 2021, pp. 5774–5783.

[26] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020.

[27] A. Navon, A. Shamsian, G. Chechik, and E. Fetaya, “Learning the pareto front with hypernetworks,” *arXiv preprint arXiv:2010.04104*, 2020.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv:1606.01540*, 2016.

[29] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour *et al.*, “Policy gradient methods for reinforcement learning with function approximation.” in *NIPs*, vol. 99. Citeseer, 1999, pp. 1057–1063.

[30] K. W. Cobbe, J. Hilton, O. Klimov, and J. Schulman, “Phasic policy gradient,” in *ICML*, 2021, pp. 2020–2027.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv:1707.06347*, 2017.

[32] G. Ostrovski, P. S. Castro, and W. Dabney, “The difficulty of passive learning in deep reinforcement learning,” *arXiv:2110.14020*, 2021.

[33] E. Bengio, J. Pineau, and D. Precup, “Interference and generalization in temporal difference learning,” in *ICML*, 2020, pp. 767–777.

[34] A. Gretton, K. Fukumizu, C. H. Teo, L. Song, B. Schölkopf, A. J. Smola *et al.*, “A kernel statistical test of independence.” in *Nips*, 2007.

[35] S. C. Chan, S. Fishman, J. Canny, A. Korattikara, and S. Guadarrama, “Measuring the reliability of reinforcement learning algorithms,” *arXiv:1912.05663*, 2019.