Revisiting Peng's Q(λ) for Modern Reinforcement Learning

Tadashi Kozuno [1 *] **Yunhao Tang** [2 *] **Mark Rowland** [3] **Rémi Munos** [4] **Steven Kapturowski** [3] **Will Dabney** [3] **Michal Valko** [4] **David Abel** [3]

Abstract

Off-policy multi-step reinforcement learning algorithms consist of conservative and non-conservative algorithms: the former actively cut traces, whereas the latter do not. Recently, Munos et al. (2016) proved the convergence of conservative algorithms to an optimal Q-function. In contrast, non-conservative algorithms are thought to be unsafe and have a limited or no theoretical guarantee. Nonetheless, recent studies have shown that non-conservative algorithms empirically outperform conservative ones. Motivated by the empirical results and the lack of theory, we carry out theoretical analyses of Peng's Q(λ), a representative example of non-conservative algorithms. We prove that *it also converges to an optimal policy* provided that the behavior policy slowly tracks a greedy policy in a way similar to conservative policy iteration. Such a result has been conjectured to be true but has not been proven. We also experiment with Peng's Q(λ) in complex continuous control tasks, confirming that Peng's Q(λ) often outperforms conservative algorithms despite its simplicity. These results indicate that Peng's Q(λ), which was thought to be unsafe, is a theoretically-sound and practically effective algorithm.

1. Introduction

Q-learning is a canonical algorithm in reinforcement learning (RL) (Watkins, 1989). It is a single-step algorithm, in that it only uses individual transitions to update value estimates. Many *multi-step* generalisations of Q-learning have been proposed, which allow temporally-extended trajectories to be used in the updating of values (Bertsekas & Ioffe, 1996; Watkins, 1989; Peng & Williams, 1994; 1996; Precup et al., 2000; Harutyunyan et al., 2016; Munos et al., 2016; Rowland et al., 2020), potentially leading to more efficient credit assignment. Indeed, multi-step algorithms have often been observed to outperform single-step algorithms for control in a variety of RL tasks (Mousavi et al., 2017; Harb & Precup, 2017; Hessel et al., 2018; Barth-Maron et al., 2018; Kapturowski et al., 2018; Daley & Amato, 2019).

However, using multi-step algorithms for RL comes with both theoretical and practical difficulties. The discrepancy between the policy that generated the data to be learnt from (the *behavior policy*) and the policy being learnt about (the *target policy*) can lead to complex, non-convergent behavior in these algorithms, and so must be considered carefully. There are two main approaches to deal with this discrepancy (cf. Table 1). *Conservative methods* ensure convergence is guaranteed no matter what behavior policy is used, typically by truncating the trajectories used for learning. By contrast, *non-conservative methods* typically do not truncate trajectories, and as a result do not come with generic convergence guarantees. Nevertheless, non-conservative methods have consistently been found to outperform conservative methods in practical large-scale applications. Thus, there is a clear gap in our understanding about non-conservative methods; why do they so work well in practice, but lack the guarantees of their conservative counterparts?

In this paper, we address this question by studying a representative non-conservative algorithm, Peng's Q(λ) (Peng & Williams, 1994; 1996, PQL), in more realistic learning settings. Our results show that while PQL does not learn optimal policies under arbitrary behavior policies, a convergence guarantee can be recovered if the behavior policy tracks the target policy, as is often the case in practice. This represents a closing of the gap between the strong empirical performance of non-conservative methods and their previous lack of theoretical guarantees.

More concretely, our primary theoretical contributions bring new understanding to PQL, and are summarized as follows:

- A proof that PQL with a *fixed* behavior policy converges to a "biased" (i.e., different from Q^*) fixed-point.
- Analysis of the quality of the resulting policy.

[*]Equal contribution [1]Independent Researcher, Okayama, Japan (Now at the University of Alberta) [2]Columbia University, NY, USA [3]DeepMind, London, UK [4]DeepMind, Paris, France. Correspondence to: Tadashi Kozuno <tadashi.kozuno@gmail.com>, Yunhao Tang <yt2541@columbia.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

Table 1. List of off-policy multi-step algorithms for control. Harutyunyan's Q(λ), Tree-backup, Watkins' Q(λ), and Peng's Q(λ) are abbreviated as HQL, TBL, WQL, and PQL, respectively (cf. Section 3.2 for details of the algorithms). Conservative column indicates if an algorithm is conservative or not (cf. Section 4). Convergence column indicates the convergence of algorithms to any fixed point, whereas Convergence to Q^* column indicates the convergence of algorithms to the optimal Q-function Q^*. ✓ indicates new results in the present paper. PQL converges to a biased fixed-point when the behavior policy is fixed. It converges to Q^* when a behavior policy is updated appropriately. (An exact condition is given in Section 5.)

Algorithm	Conservative	Convergence	Convergence to Q^*
α-TRACE (ROWLAND ET AL., 2020)	NO	?	?
C-TRACE (ROWLAND ET AL., 2020)	NO	?	?
HQL (HARUTYUNYAN ET AL., 2016)	NO	✓(WITH SMALL λ)	✓(WITH SMALL λ)
RETRACE (MUNOS ET AL., 2016)	YES	✓	✓
TBL (PRECUP ET AL., 2000)	YES	✓	✓
UNCORRECTED n-STEP RETURN	NO	?	?
WQL (WATKINS, 1989)	YES	✓	✓
PQL (PENG & WILLIAMS, 1994)	NO	✓ (BIASED)	✓ (CF. CAPTION)

- Convergence of PQL to an optimal policy when using appropriate behavior policy updates.
- Error propagation analysis when using approximations.

In addition to these theoretical insights, we validate the empirical performance of PQL through extensive experiments. Our focus is on continuous control tasks, where one encounters many technical challenges that do not exist in discrete control tasks (cf. Section 7.2). They are also accessible to a wider range of readers. We show that PQL can be easily extended to popular off-policy actor-critic algorithms such as DDPG, TD3 and SAC (Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018). Over a large subset of tasks, PQL consistently outperforms other conservative and non-conservative baseline alternatives.

2. Notation and Definitions

For a finite set $\mathbf{A}$ and an arbitrary set $\mathbf{B}$, we let $\Delta_{\mathbf{A}}$ and $\mathbf{B}^{\mathbf{A}}$ be the probability simplex over $\mathbf{A}$ and the set of all mappings from $\mathbf{A}$ to $\mathbf{B}$, respectively.

Markov Decision Processes (MDP). We consider an MDP defined by a tuple $\langle \mathbf{X}, \mathbf{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma \rangle$, where $\mathbf{X}$ is the finite state space, $\mathbf{A}$ the finite action space, $\mathcal{P} : \mathbf{X} \times \mathbf{A} \to \Delta_{\mathbf{X}}$ the state transition probability kernel, $\mathcal{P}_0 \in \Delta_{\mathbf{X}}$ the initial state distribution, $\mathcal{R}$ the (conditional) reward distribution, and $\gamma \in [0, 1)$ the discount factor (Puterman, 1994). We let $r \in \mathbb{R}^{\mathbf{X} \times \mathbf{A}}$ be a reward function defined by $r(x, a) := \int r' \mathcal{R}(\mathrm{d}r'|x, a)$.

On the Finiteness of the State and Action Spaces. While we assume both $\mathbf{X}$ and $\mathbf{A}$ to be finite, most of theoretical results in the paper hold in continuous state spaces with appropriate measure-theoretic considerations. The finiteness assumption on the action space is necessary to guarantee the existence of the optimal policy (Puterman, 1994). In Appendix B, we discuss assumptions necessary to extend our theoretical results to continuous action spaces.

Policy and Value Functions. Suppose a policy $\pi : \mathbf{X} \to \Delta_{\mathbf{A}}$. We consider the standard RL setup where an agent interacts with an environment, generating a sequence of state-action-reward tuples $(X_t, A_t, R_t)_{t \geq 0}$ with A_t being an action sampled from some policy; throughout, we denote random variables by upper cases. Define $G = \sum_{t=0}^{\infty} \gamma^t R_t$ as the cumulative return. The state-value and Q-functions are defined by $V^\pi(x) := \mathbb{E}[G|X_0 = x, \pi]$ and $Q^\pi(x, a) := \mathbb{E}[G|X_0 = x, A_0 = a, \pi]$, respectively, where the conditioning by π means $A_t \sim \pi(\cdot|X_t)$.

Evaluation and Control. Two key tasks in RL are evaluation and control. The problem of evaluation is to learn the Q-function of a fixed policy. The aim in the control setting is to learn an optimal policy π_* defined as to satisfy $V^{\pi_*} := V^* \geq V^\pi, \forall \pi$ (the inequality is point-wise, i.e., $V^*(x) \geq V^\pi(x)$ for all $x \in \mathbf{X}$). Similarly to V^*, we let Q^* denote the optimal Q-function Q^{π_*}. As a greedy policy with respect to Q^* is optimal, it suffices to learn Q^*. In this paper, we are particularly interested in the off-policy control setting, where an agent collects data with a behavior policy μ, which is not necessarily the agent's current policy π. On-policy settings are a special case where $\pi = \mu$.

3. Multi-step RL Algorithms and Operators

Operators play a crucial role in RL since all value-based RL algorithms (exactly or approximately) update a Q-function based on the recursion $Q_{k+1} := \mathcal{O}_k Q_k$, where $\mathcal{O}_k : \mathbb{R}^{\mathbf{X} \times \mathbf{A}} \to \mathbb{R}^{\mathbf{X} \times \mathbf{A}}$ is an operator that characterizes

each algorithm. In this section, we review multi-step RL algorithms and their operators.

Basic Operators. Assume we have a fixed policy π. With an abuse of notations, we define operators $\pi : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}}$ and $\mathcal{P} : \mathbb{R}^{\mathbf{X}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ by

$$(\pi Q)(x) := \textstyle\sum_{a\in\mathbf{A}} \pi(a|x)Q(x,a), \text{ and}$$
$$(\mathcal{P}V)(x,a) := \textstyle\sum_{y\in\mathbf{X}} \mathcal{P}(y|x,a)V(y)$$

for any $Q \in \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ and $V \in \mathbb{R}^{\mathbf{X}}$, respectively (hereafter, we omit "for any..." in definitions of operators for brevity). We define their composite $\mathcal{P}^\pi := \mathcal{P}\pi$. As a result, the Bellman operator $\mathcal{T}^\pi : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ is defined by $\mathcal{T}^\pi Q := r + \gamma\mathcal{P}^\pi Q$. For a function $Q \in \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$, we let $\mathbf{G}(Q)$ be the set of all greedy policies[1] with respect to Q. The Bellman optimality operator $\mathcal{T}$ is defined by $\mathcal{T}Q = \mathcal{T}^{\pi_Q}Q$ with $\pi_Q \in \mathbf{G}(Q)$[2]. Q-learning approximates the value iteration (VI) updates $Q_{k+1} := \mathcal{T}Q_k$.

3.1. On-policy Multi-step Operators for Control

We first introduce on-policy multi-step operators for control.

Modified Policy Iteration (MPI). MPI uses the recursion $Q_{k+1} := \mathcal{T}_n^{\pi_k}Q_k$ for Q-function updates (Puterman & Shin, 1978), where $\pi_k \in \mathbf{G}(Q_k)$. The n-step return operator $\mathcal{T}_n^\pi : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ is defined by $\mathcal{T}_n^\pi Q := (\mathcal{T}^\pi)^n Q$.

λ-Policy Iteration (λ-PI). λ-PI uses the recursion $Q_{k+1} := \mathcal{T}_\lambda^{\pi_k}Q_k$ for Q-function updates (Bertsekas & Ioffe, 1996), where $\pi_k \in \mathbf{G}(Q_k)$. The λ-return operator $\mathcal{T}_\lambda^\pi : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ is defined as

$$\begin{aligned}\mathcal{T}_\lambda^\pi Q &:= (1-\lambda)\textstyle\sum_{n=1}^\infty \lambda^{n-1}\mathcal{T}_n^\pi Q \\ &= Q + (\mathcal{I} - \gamma\lambda\mathcal{P}^\pi)^{-1}(\mathcal{T}^\pi Q - Q),\end{aligned}$$

where $(\mathcal{I} - \gamma\lambda\mathcal{P}^\pi)^{-1} := \sum_{t=0}^\infty (\gamma\lambda\mathcal{P}^\pi)^t$, and $\lambda \in [0,1]$.

3.2. Off-policy Multi-step Operators for Control

Next, we explain off-policy multi-step operators for control. We note that on-policy algorithms in the last subsection can be converted to off-policy versions by using importance sampling (Precup et al., 2000; Casella & Berger, 2002).

Uncorrected n-step Return. For a sequence of behavior policies $(\mu)_{k\geq 0}$, the uncorrected n-step return algorithm uses the recursion $Q_{k+1} := \mathcal{N}_n^{\mu_k,\pi_k}Q_k$ for Q-function updates (Hessel et al., 2018; Kapturowski et al., 2018), where $\pi_k \in \mathbf{G}(Q_k)$. Here, the uncorrected n-step return operator $\mathcal{N}_n^{\mu,\pi}$ is defined for any policies π and μ by

$$\mathcal{N}_n^{\mu,\pi}Q := (\mathcal{T}^\mu)^{n-1}\mathcal{T}^\pi Q.$$

[1]Note that there may be multiple greedy policies due to ties.

[2]Note that this definition is independent of the choice of π_Q.

Peng's Q(λ) (PQL) For a sequence of behavior policies $(\mu)_{k\geq 0}$, PQL uses the recursion $Q_{k+1} := \mathcal{N}_\lambda^{\mu_k,\pi_k}Q_k$ for Q-function updates (Peng & Williams, 1994; 1996), where $\pi_k \in \mathbf{G}(Q_k)$. Here, the PQL operator $\mathcal{N}_\lambda^{\mu,\pi}$ is defined for any policies π and μ by

$$\mathcal{N}_\lambda^{\mu,\pi}Q := (1-\lambda)\textstyle\sum_{n=1}^\infty \lambda^{n-1}\mathcal{N}_n^{\mu,\pi}Q\,, \qquad (1)$$

where $\lambda \in [0,1]$. Note that *PQL is a generalization of λ-PI* because it reduces to λ-PI when $\mu_k = \pi_k$. In other words, PQL is λ-PI with one additional degree of freedom in μ_k.

General Retrace. We next introduce a general version of the Retrace operator (Munos et al., 2016), from which other operators are obtained as special cases.

For a behavior policy μ and a target policy π, we let $\mathcal{P}^{c\mu} : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ be an operator defined by

$$(\mathcal{P}^{c\mu}Q)(x,a) := \sum_{(y,b)\in\mathbf{X}\times\mathbf{A}} \mathcal{P}(y|x,a)c(y,b)\mu(b|y)Q(y,b),$$

where c is an arbitrary non-negative function over $\mathbf{X}\times\mathbf{A}$ whose choice depends on an algorithm. Note that for any n, $((\mathcal{P}^{c\mu})^n Q)(x,a)$ can be estimated off-policy with data collected under the behavior policy μ.

A general Retrace operator $\mathcal{R}_\lambda^{c\mu,\pi} : \mathbb{R}^{\mathbf{X}\times\mathbf{A}} \to \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ is obtained by replacing $\mathcal{P}^\pi$ of $(\mathcal{I} - \gamma\lambda\mathcal{P}^\pi)^{-1}$ in the λ-return operator $\mathcal{T}_\lambda^\pi$ with $\mathcal{P}^{c\mu}$. Concretely,

$$\mathcal{R}_\lambda^{c\mu,\pi}Q := Q + (\mathcal{I} - \gamma\lambda\mathcal{P}^{c\mu})^{-1}(\mathcal{T}^\pi Q - Q).$$

The general Retrace algorithm updates its Q-function by $Q_{k+1} := \mathcal{R}_\lambda^{c_k\mu_k,\pi_k}Q_k$, where $(c_k)_{k\geq 0}$ is a sequence of arbitrary non-negative functions over $\mathbf{X}\times\mathbf{A}$, $(\mu_k)_{k\geq 0}$ is an arbitrary sequence of behavior policies, and $(\pi_k)_{k\geq 0}$ is a sequence of target policies that depends on an algorithm. Given the choices of c_k and π_k in Table 2, we recover a few known algorithms (Watkins, 1989; Peng & Williams, 1994; 1996; Precup et al., 2000; Harutyunyan et al., 2016; Munos et al., 2016; Rowland et al., 2020).

The general Retrace algorithm is off-policy as $(\mathcal{R}_\lambda^{c_k\mu_k,\pi_k}Q_k)(x_0,a_0)$ can be estimated off-policy by the following estimator given a trajectory $(x_t,a_t,r_t)_{t\geq 0}$ collected under μ_k:

$$Q_k(x_0,a_0) + \sum_{t=0}^\infty \Big(\prod_{u=1}^t c(x_u,a_u)\Big)\gamma^t\lambda^t\delta_t, \qquad (2)$$

where $\prod_{u=1}^0 c(x_u,a_u) := 1$, and δ_t is the TD error $r_t + \gamma(\pi_k Q_k)(x_{t+1}) - Q_k(x_t,a_t)$ at time step t.

4. Conservative and Non-conservative Multi-step RL Algorithms

Munos et al. (2016) showed that the following conditions suffice for the convergence of the general Retrace to Q^*:

Table 2. Choices of c_k and π_k in off-policy multi-step operators for control. See Section 3.2 for details. The same abbreviations as those in Table 1 are used. For brevity, we defined $\pi_{Q_k} \in \mathbf{G}(Q_k)$. We denote $\pi_k(a|x)/\mu_k(a|x)$ by $\rho_k(x,a)$ and $(1-\alpha)+\alpha\pi_{Q_k}(a|x)/\mu_k(a|x)$ by $\widetilde{\rho}_k(x,a)$. α-trace and C-trace look the same in the table, but C-trace adaptively changes α so that the trace length matches to a target trace length.

Algorithm	c_k	π_k
α-TRACE	$\min\{1, \widetilde{\rho}_k\}$	$\alpha\pi_{Q_k} + (1-\alpha)\mu_k$
C-TRACE	$\min\{1, \widetilde{\rho}_k\}$	$\alpha\pi_{Q_k} + (1-\alpha)\mu_k$
HQL	1	π_{Q_k}
RETRACE	$\min\{1, \rho_k\}$	ANY
TBL	π_k	ANY
WQL	$\min\{1, \rho_k\}$	π_{Q_k}
PQL	1	$\lambda\pi_{Q_k} + (1-\lambda)\mu_k$

1. $c_k(x,a) \in [0, \pi_k(a|x)/\mu_k(a|x)]$ for any k and $(x,a) \in \mathbf{X}\times\mathbf{A}$.
2. π_k satisfies some greediness condition, such as ε-greediness with decreasing ε as k increases; cf. Munos et al. (2016) for further details.

We call algorithms that satisfy the first condition *conservative* algorithms for reasons to be explained below. Otherwise, we call the algorithms *non-conservative*. See Table 1 for the classification of algorithms. The uncorrected n-step return algorithm can also be viewed as a non-conservative algorithm with *non-Markovian traces* that depend also on the past.

Conservativeness, Theoretical Guarantees, and Empirical Performance of Algorithms. Recall that in the general Retrace update estimator (2), the effect of the TD error δ_t is attenuated by $\prod_{u=1}^{t} c(x_u, a_u)$ in addition to $\gamma^t\lambda^t$. Hence, from the backward view (Sutton & Barto, 1998), the first condition intuitively requires that *the trace must be cut* if a sub-trajectory $(x_0, a_0, \ldots, x_t, a_t)$ is unlikely under π_k relative to μ_k. As a result, conservative algorithms only carry out *safe* updates to Q-functions.

As shown in (Munos et al., 2016), such conservative updates enable a convergence guarantee of general conservative algorithms. However, Rowland et al. (2020) observed that it often results in frequent trace cuts, and conservative algorithms usually benefit less from multi-step updates.

In contrast, non-conservative algorithms accumulate TD errors without carefully cutting traces. As a result, non-conservative algorithms might perform poorly. As we show later (Proposition 5), it is the case at least for Harutyunyan's Q(λ) (Harutyunyan et al. (2016), HQL), an instance of non-conservative algorithms, when a behavior policy is fixed. Nonetheless, non-conservative algorithms are known to perform well in practice (Hessel et al., 2018; Kapturowski et al., 2018; Daley & Amato, 2019). To understand its reason, it is important to characterize what kind of updates to the behavior policy entail the convergence of the overall algorithm. In the following sections, we take a step forward along this direction. We establish the convergence guarantee of PQL under two setups: **(1)** when the behavior policy is fixed; **(2)** when the behavior policy is updated in an appropriate way.

5. Theoretical Analysis of Peng's Q(λ)

In this section, we analyze Peng's Q(λ). We start with the *exact case* where there is no update errors in value functions. Later, we will consider the *approximate case* when accounting for update errors. The following lemma is particularly useful in theoretical analyses as well as practical implementations.

Lemma 1 (Harutyunyan et al., 2016)**.** *The PQL operator can be rewritten in the following forms:*

$$\mathcal{N}_\lambda^{\mu,\pi} Q = Q + (\mathcal{I} - \gamma\lambda\mathcal{P}^\mu)^{-1}\Big(\mathcal{T}^{\lambda\mu+(1-\lambda)\pi}Q - Q\Big)$$
$$= (\mathcal{I} - \gamma\lambda\mathcal{P}^\mu)^{-1}(r + \gamma(1-\lambda)\mathcal{P}^\pi Q)\,.$$

Proof. This is proven in (Harutyunyan et al., 2016), but we provided a proof in Appendix C for completeness. □

5.1. Exact Case with a Fixed Behavior Policy

We now analyze PQL with a fixed behavior policy μ. While the behavior policy is not fixed in a practical situation, the analysis shows a trade-off between bias and convergence rate. This trade-off is analogous to the bias-contraction-rate trade-off of off-policy multi-step algorithms for policy evaluation (Rowland et al., 2020) and sheds some light on important properties of PQL.

Concretely, we analyze the following algorithm:

$$\pi_k \in \mathbf{G}(Q_k) \text{ and } Q_{k+1} := \mathcal{N}_\lambda^{\mu,\pi_k} Q_k. \tag{3}$$

Harutyunyan et al. (2016) has proven that a fixed point of the PQL operator coincides with the unique fixed point of $\lambda\mathcal{T}^\mu + (1-\lambda)\mathcal{T}$, which is guaranteed to exist since $\lambda\mathcal{T}^\mu + (1-\lambda)\mathcal{T}$ is a contraction with modulus γ under L^∞-norm (see Appendix A for details about the contraction and other notions).

The existence of a fixed point does not imply the convergence of PQL, and we need to show that the distance between Q_k and the fixed point is decreasing. With the following theorem, we show that PQL does converge.

Theorem 2. *Let $\pi_\dagger$ be a policy such that $Q^{\lambda\mu+(1-\lambda)\pi_\dagger} \geq Q^{\lambda\mu+(1-\lambda)\pi}$ for any policy π, where the inequality is pointwise. Then, $\pi_\dagger \in \mathbf{G}(Q^{\lambda\mu+(1-\lambda)\pi_\dagger})$, and Q_k of PQL (3) uniformly converges to $Q^{\lambda\mu+(1-\lambda)\pi_\dagger}$ with the rate β^k, where $\beta := \gamma(1-\lambda)/(1-\gamma\lambda)$.*

Proof. See Appendix E. □

We build intuitions about the bias-convergence-rate trade-off implied in Theorem 2. When λ increases, the fixed point is $Q^{\lambda\mu+(1-\lambda)\pi_\dagger}$, whose bias against Q^* arguably increases; at the same time, the contraction rate β decreases, so that the contraction is faster.

Remark 1. *In Section 7.6 of (Sutton & Barto, 1998), it is conjectured that PQL with a fixed policy would converge to a hybrid of Q^μ and Q^*. Theorem 2 gives an answer to this conjecture and shows that Sutton & Barto (1998)'s conjecture is not necessarily true. Rather, the theorem shows that PQL converges to the Q-function of the best policy among policies of the form $\lambda\mu + (1-\lambda)\pi$.*

5.2. Approximate Case with a Fixed Behavior Policy

In practice, value-update errors are inevitable due to e.g., finite-sample estimations and function approximation errors. In this subsection, we provide the error propagation analysis of PQL with a fixed behavior policy. As we will see, the analysis depicts a trade-off between fixed point bias and error tolerance.

We analyze the following algorithm:

$$\pi_k \in \mathbf{G}(Q_k) \text{ and } Q_{k+1} := \mathcal{N}_\lambda^{\mu,\pi_k} Q_k + \varepsilon_k \,,$$

where $\varepsilon_k \in \mathbb{R}^{\mathbf{X}\times\mathbf{A}}$ denotes the value-update error at iteration k. For simplicity, we use $\rho_k := \lambda\mu + (1-\lambda)\pi_k$ and $\rho_\dagger := \lambda\mu + (1-\lambda)\pi_\dagger$ in this subsection.

Remark 2. *We emphasize that ε_k should be rather understood as $Q_{k+1} - \mathcal{N}_\lambda^{\mu,\pi_k} Q_k$,the difference between the function Q_{k+1} at $k+1$-th iteration and the ideal update $\mathcal{N}_\lambda^{\mu,\pi_k} Q_k$. In practice, Q_{k+1} is obtained by first constructing a sample estimate $\mathcal{N}_\lambda^{\mu,\pi_k} Q_k$ and then fitting a parametric model to it by, for example, the square loss minimization. As a result, ε_k typically consists of estimation error and function approximation error.*

In Section 5.1, we showed $\lim_{k\to\infty} Q_k = Q^{\lambda\mu+(1-\lambda)\pi_\dagger}$ when $\varepsilon_k(x,a) = 0$ at every $(x,a) \in \mathbf{X}\times\mathbf{A}$, and $\pi_\dagger \in \mathbf{G}(Q^{\lambda\mu+(1-\lambda)\pi_\dagger})$. Therefore, π_k is an approximation to $\pi_\dagger$, and thus it is natural to define $V^{\rho_\dagger} - V^{\rho_k}$ as the loss of using π_k rather than $\pi_\dagger$. The following theorem provides an upper bound for the loss.

Theorem 3. *For any K, the following holds:*

$$\|V^{\rho_\dagger} - V^{\rho_K}\|_\infty \le O(\beta^K) + \frac{2}{1-\gamma}\sum_{k=0}^{K-1} \beta^{K-k-1}\|\varepsilon_k\|_\infty \,,$$

where $\|\cdot\|_\infty$ is the L_∞-norm defined for any real-valued function f by $\|f\|_\infty := \max_v |f(v)|$.

Proof. See Appendix G. □

Remark 3. *In Theorem 3, we provide an upper bound of the L_∞-norm of $V^{\rho_\dagger} - V^{\rho_K}$. As a result, the L_∞-norm of ε_k appears in the upper-bound. However in Appendix G, we prove a point-wise upper-bound of $V^{\rho_\dagger} - V^{\rho_K}$, from which an L_p-norm upper-bound can be obtained in a straightforward way (e.g., see Lemma 6 of Scherrer et al. (2015)). For simplicity, we present the L_∞-norm upper-bound.*

As we have already explained the bias-convergence-rate trade-off, for now we ignore the $O(\beta^K)$ term and focus on the error term. For simplicity, we assume $\|\varepsilon_k\|_\infty = \varepsilon$ for every k. Then,

$$\frac{2}{1-\gamma}\sum_{k=0}^{K-1} \beta^{K-k-1}\|\varepsilon_k\|_\infty = O\left(\frac{1-\gamma\lambda}{(1-\gamma)^2}\varepsilon\right),$$

In contrast, an analogous result of λ-PI is $O(\varepsilon/(1-\gamma)^2)$ (Scherrer, 2013). When $\lambda = 0$, these results coincide, which is expected since both λ-PI and PQL degenerate to value iteration. When $\lambda = 1$, PQL's error dependency is $O(\varepsilon/(1-\gamma))$, which is significantly better than $O(\varepsilon/(1-\gamma)^2)$. However in this case, PQL is completely biased and converges to Q^μ. At intermediate values of λ, PQL achieves a trade-off between error tolerance with bias by changing λ.

5.3. Approximate Case with Behavior Policy Updates

Previously, we have analyzed PQL with a fixed behavior policy. However, in practice, the behavior policy is updated along with the target policy. Besides, value-update errors are inevitable in complex tasks. As a result, PQL may behave quite differently in a practical scenario. This motivates our analysis for the following algorithm:[3]

$$\begin{aligned} Q_{k+1} &:= \mathcal{N}_\lambda^{\mu_k,\pi_k} Q_k + \varepsilon_k \\ \mu_k &:= \alpha\pi_k + (1-\alpha)\mu_{k-1}, \end{aligned} \tag{4}$$

where $\pi_k \in \mathbf{G}(Q_k)$, and $\alpha \in [1-\lambda, 1]$. Note that when $\alpha = 1$, this algorithm reduces to λ-PI as a special case. Though this behavior policy update closely resembles to that of conservative policy iteration (Kakade & Langford, 2002), here we require $\alpha \ge 1-\lambda$.

This algorithm has the following performance guarantee.

Theorem 4. *For any K, the following holds:*

$$\|V^* - V^{\pi_K}\|_\infty \le O(\zeta^K) + \frac{2}{1-\gamma}\sum_{l=0}^{K-1} \zeta^{K-l-1}\|\varepsilon_l\|_\infty \,,$$

where $\zeta := 1-\alpha+\alpha\gamma$. Hence, PQL with behavior policy updates converges to the optimal policy with the rate ζ^K.

[3] This algorithm updates the behavior policy after each application of the PQL operator. In Appendix F, we analyze a case where the behavior policy is updated after multiple applications of the PQL operator.

Proof. See Appendix H. □

Remark 4. *As noted in Remark 3, we can derive an L_p-norm upper-bound of $V^* - V^{\pi_K}$ given its point-wise upper-bound. In Appendix H, we actually provide its point-wise upper-bound.*

Remark 5. *Our results differ from existing work on off-policy learning and multi-step methods in several important respects. Munos et al. (2016) also analyse multi-step off-policy algorithms, but focus on identifying conditions for algorithms to be* conservative*, and provide asymptotic convergence guarantees of Q_k in the tabular setting under* any *choice of behaviour policies. In contrast, our analysis focuses on* non-conservative *algorithms, and provides a finite-time error bound for $V^* - V^{\pi_k}$ that incorporates approximation in the algorithm steps. In this regard, these results bear similarity with earlier analysis of multi-step* on-policy *algorithms such as λ-policy iteration (Scherrer, 2013). The central distinction is that our analysis clarifies what conditions are sufficient in non-conservative off-policy algorithms to retain convergence of V^{π_k} to V^*.*

The first term on the right hand side shows the convergence of PQL with behavior policy updates in an exact case, i.e., $\|\varepsilon_k\|_\infty = 0$ for any k. It states that the fastest convergence rate is γ^K (achieved when $\alpha = 1$), which is the same as the convergence rate of VI (Munos, 2005), policy iteration (Munos, 2003), MPI (Scherrer et al., 2012; 2015), and λ-PI (Scherrer, 2013). When $\alpha \neq 1$, the convergence rate coincides with that of conservative policy iteration (Scherrer, 2014). However we are not aware of a similar result of conservative λ-PI, which would be an analogue of PQL considered here. Theorem 4 also provides the error dependency of PQL (the second term on the right hand side). It coincides with the previous result of the above algorithms when $\alpha = 1$, as one would expect, since PQL with $\alpha = 1$ is precisely λ-PI. Nonetheless PQL allows some degree of off-policiness when $\alpha \neq 1$.

5.4. Oscillatory Behavior of HQL

In this section, we have proven the convergence of exact PQL (i.e., no value-update errors). However, the following proposition shows that exact HQL, an instance of non-conservative algorithms, does not converge in an MDP when the behavior policy is fixed. Nonetheless, in the same MDP, setting the behavior policy μ_k to a greedy policy $\pi_k \in \mathbf{G}(Q_k)$ guarantees the convergence.

Proposition 5. *There is an MDP such that when exact HQL is run with a fixed policy $\mu_k = \mu$ for all k, $\lambda = 1$, and $Q_0 = Q^\mu$, HQL's Q-function Q_k oscillates between two functions, and its greedy policy π_k oscillate between optimal and sub-optimal policies. Contrarily, if $\mu_k \in \mathbf{G}(Q_k)$, HQL converges to an optimal policy.*

Proof. A proof of the first claim is given in Appendix D. The second claim immediately follows by noting that if $\mu_k = \pi_k \in \mathbf{G}(Q_k)$, HQL is λ-PI, which is known to converge (Bertsekas & Ioffe, 1996). □

While this result is specialized to HQL, it sheds light on an important aspect of non-conservative algorithms in general:

While non-conservative algorithms may perform poorly when the behavior policy is fixed, they may converge to Q^* when the behavior policy is updated.

The above captures a critical aspect of how algorithms behave in practice, where the behavior policy is continuously updated.

6. Deep RL Implementations

We next show that Peng's Q(λ) can be conveniently implemented with established off-policy deep RL algorithms. Our experiments focus on continuous control problems where the action space $\mathbf{A} = [-1, 1]^m$. A primary motivation for considering continuous control benchmarks (e.g., (Brockman et al., 2016; Tassa et al., 2020)) is that they are usually more accessible to a wider RL research community, compared to challenging discrete control benchmarks such as Atari games (Bellemare et al., 2013).

6.1. Off-policy Actor-critic Algorithms

Off-policy actor-critic algorithms maintain a policy $\pi_\theta(a|x)$ with parameter θ and a Q-function critic $Q_\phi(x, a)$ with parameter ϕ. For the policy, a popular choice is the point mass distribution $\pi_\theta(a|x) = \delta(a - \pi_\theta(x))$, where $\pi_\theta(x) \in \mathbb{R}^{\mathbf{A}}$ (Lillicrap et al., 2016; Fujimoto et al., 2018; Barth-Maron et al., 2018). The algorithm collects data with an exploratory behavior policy μ and saves tuples (x_t, a_t, r_t) into a replay buffer $\mathcal{D}$. At each training iteration, the critic $Q_\phi(x, a)$ is updated by minimizing squared errors against a Q-function target $\mathbb{E}_D\left[(Q_\phi(x, a) - Q_{\text{target}}(x, a))^2\right]$. The policy is updated via the deterministic policy gradient $\theta \leftarrow \theta + \alpha \mathbb{E}_\mu\left[\nabla_\theta Q_\phi(x, \pi_\theta(x))\right]$ (Silver et al., 2014). See further details in Appendix J.

6.2. Implementations of Multi-step Operators

While approximate estimates to $\mathcal{T}Q(x, a)$ are arguably the simplest to implement, it only myopically looks ahead for one step. Usually, the learning can be significantly sped up when the targets are constructed with multi-step operators. (See, e.g, empirical examples in (Hessel et al., 2018; Barth-Maron et al., 2018; Kapturowski et al., 2018) and theoretical insights in (Rowland et al., 2020)) For example, the uncorrected n-step operator is estimated as follows (Hessel et al., 2018): given a n-step trajectory $(x_i, a_i, r_i)_{i=0}^n$,

the target at (x_0, a_0) is computed as $Q_{\text{target}}(x_0, a_0) = \sum_{i=0}^{n-1} \gamma^i r_0 + \gamma^n Q_{\phi^-}(x_n, \pi_{\theta^-}(x_n))$. Similar estimates could be derived for all multi-step operators introduced in Section 3, especially Peng's Q(λ). We present full details in Appendix J.

Desirable empirical properties of Peng's Q(λ). The estimates of Peng's Q(λ) do not require importance sampling ratios $\frac{\pi(a|x)}{\mu(a|x)}$. This is especially valuable for continuous control, where the policy could be deterministic, in which case algorithms such as Retrace (Munos et al., 2016) cuts traces immediately. Even when policies are stochastic and traces based on IS ratios are not cut immediately, prior work suggests that the trace cuts are usually pessimistic especially for high-dimensional action space (see, e.g., (Wang et al., 2017) for implementation techniques to mitigate the issue).

7. Experiments

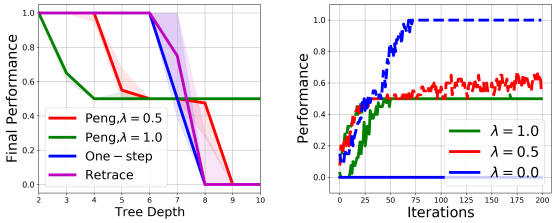
To build better intuitions about Peng's Q(λ), we start with tabular examples in Section 7.1. We will see that the empirical properties of Peng's Q(λ) echo the theoretical analysis in previous sections. In Section 7.2, we evaluate Peng's Q(λ) in the deep RL contexts. We combine Peng's Q(λ) with baseline deep RL algorithms and compare its performance against alternative operators.

7.1. A tabular example

Tree MDP. We consider toy examples with a tree MDP of depth D. The MDPs are binary trees, with each node corresponding to a state. Starting from any non-leaf state, the two actions $a \in \{L, R\}$ transition the agent to one of its child nodes with probability one. Each episode lasts for D steps and the agent always starts at the root node. The rewards are zero everywhere except $r = 1$ at the leftmost leaf node and $r = 0.5$ at the rightmost leaf node. The behavior policy μ is $\mu(L|x) = 0.3, \mu(R|x) = 0.7$ for all states x.

Note that there is a sub-optimal policy of collecting $r = 0.5$ at the rightmost leaf. The behavior policy is by design biased towards taking right moves, such that it is easy for the agent to learn the sub-optimal policy. The optimal policy is to take left moves and collect $r = 1$. Throughout training, we optimize the target policy π while fixing the behavior policy μ. This echos the theoretical setup in Section 5.2. See Appendix J for further details on the setup.

Results. In Figure 1(a), we show the performance of different algorithms after $10,000$ iterations as a function of the MDP's tree depth D. When $D = 2$, all algorithms achieve the optimal performance; when $\lambda = 1$, as D increases, the fixed point bias of Peng's Q(λ) hurts the performance drastically. This is less severe for $\lambda = 0.5$, whose performance decays less quickly. On the other hand, both Retrace and the one-step operator learn the optimal policy even for $D \leq 6$. However, when D increases, it becomes difficult to sample the optimal trajectory. As such, the sparse rewards make it difficult to learn meaningful Q-functions in a reasonable amount of time, unless the return signals get propagated effectively (i.e,. do not cut traces). This is shown in Figure 1(a), where Peng's Q(λ) with $\lambda = 1$ is the only method that finds a policy with non-zero expected return.



(a) Final performance (b) Learning curves

Figure 1. Performance on tree MDPs. Figure(a) shows how performance (after $10,000$ iterations) changes as a function of tree depth D; Figure(b) shows the learning curves of different operators.

Similar observations are made in Figure 1(b), where we compare Peng's Q(λ) for various λ under $D = 10$ (solid lines) and $D = 5$ (dotted lines). Small λ corresponds to less bias in the Q-function fixed points and should asymptotically converge to higher performance, but its ineffective reward signal propagation hinders policy improvement in a reasonable time when D is large; on the other hand, large λ suffers sub-optimality when D is small, but its initial policy improvement is substantially expedited when the D is large.

7.2. Deep RL experiments

Evaluations. We evaluate performance over environments with a number of different physics simulation backends, such as MuJoCo (Todorov et al., 2012) based DeepMind (DM) control suite (Tassa et al., 2020) and an open sourced simulator Bullet physics (Coumans & Bai, 2016–2019). Due to space limit, below we only show results for DM control suite and provide a more complete set of evaluations in Appendix J.

Baseline comparison. We use TD3 (Fujimoto et al., 2018) as the base algorithm.[4] We compare with a few multi-step baselines: **(1)** one-step (also the base algorithm); **(2)**

[4]TD3 reasonably echoes the theoretical setup in Section 5.3: both the behavior and target policies are near-greedy policy (slowly) trying to maximize Q_k, and Q-value updates are performed by using data obtained by following the behavior policy and stored in the replay buffer.

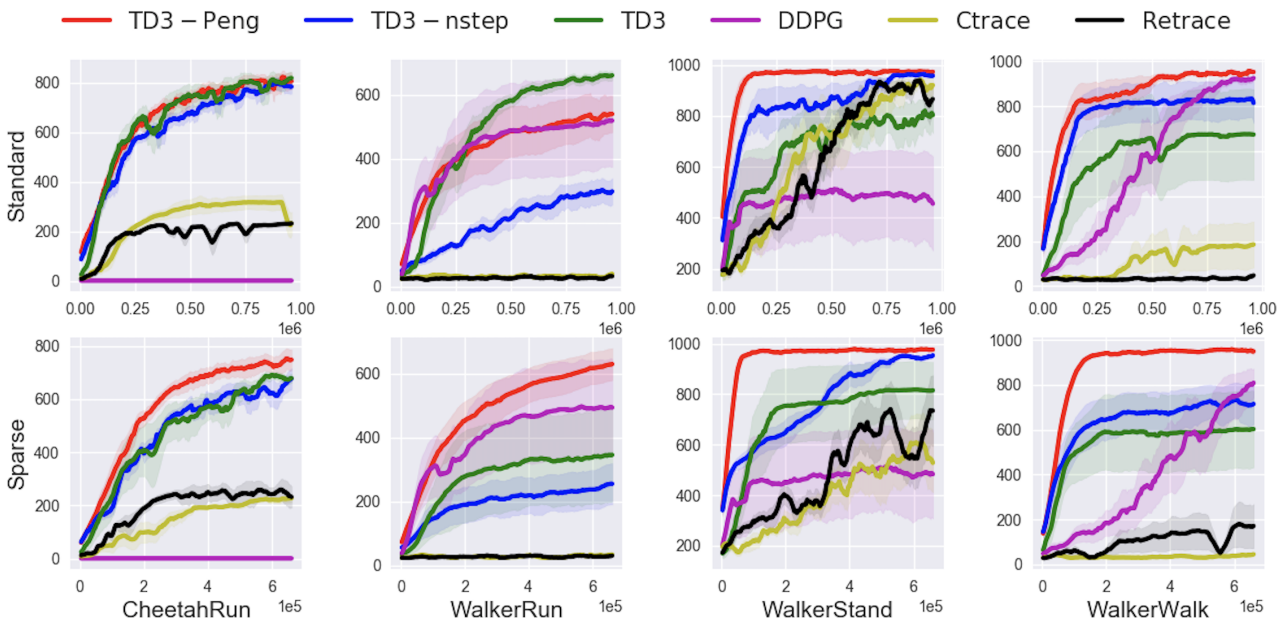


Figure 2. Evaluation of baseline algorithms over standard DM control domains. The first row shows results on standard benchmarks; the second row shows results on sparse reward variants of the benchmarks. Four task names are labeled at the bottom. In each plot, x-axis shows the number of training steps and y-axis shows the performance. In standard benchmarks, Peng's Q(λ) generally performs more stably than other algorithms; in sparse reward benchmarks, Peng's Q(λ) outperforms all other algorithms across all presented tasks.

Uncorrected n-step with a fixed n; **(3)** Peng's Q(λ) with a fixed λ; **(4)** Retrace and C-trace. Among all baselines, uncorrected n-step operator is the most commonly used non-conservative operator while Retrace is a representative conservative operator. See Appendix J for more details. All algorithms are trained with a fixed number of steps and results are averaged across 5 random seeds.

Standard benchmark results. In the top row of Figure 2, we show evaluations on standard benchmarks. Across most tasks, Peng's Q(λ) performs more stably than other baseline algorithms. We see that Peng's Q(λ) learns generally as fast as other baselines, and in some cases significantly faster than others. Note that though Peng's Q(λ) does not necessarily obtain the best learning performance *per each task*, it consistently ranks as the top two algorithms (with ties). This is in contrast to baseline algorithms whose performance rank might vary drastically across tasks. For example, the one-step TD3 performs well in CheetahRun while performs poorly in WalkerWalk. Also, both Ctrace and Retrace generally significantly perform more poorly. We provide further analysis in Appendix J.

Sparse rewards results. In the bottom row of Figure 2, we show evaluations on sparse reward variants of the benchmark tasks. See details on these environments in Appendix J. Sparse rewards are challenging for deep RL algorithms, as it is more difficult to numerically propagate learning signals across time steps. Accordingly, sparse rewards are natural benchmarks for operator-based algorithms. Across all tasks, Peng's Q(λ) consistently outperforms other baselines. In a few cases, uncorrected n-step also outperforms the baseline TD3 – we speculate that this is because the former propagates the learning signal more efficiently, which is critical for sparse rewards. Compared to uncorrected n-step, Peng's Q(λ) seems to achieve a better trade-off between efficient propagation of learning signals and fixed point biases, which leads to relatively stable and consistent performance gains across all selected benchmark tasks.

7.3. Additional deep RL experiments

Maximum-entropy RL. In Appendix I, we show how Peng's Q(λ) could be extended to maximum-entorpy RL (Ziebart et al., 2008; Fox et al., 2016; Haarnoja et al., 2017; 2018). We combine multi-step operators with maximum-entropy deep RL algorithms such as SAC (Haarnoja et al., 2018) and show performance gains over benchmark tasks. See Appendix J for further details.

Ablation study on λ. In Appendix J, we provide an ablation study on the effect of λ. We show that the performance of Peng's Q(λ) depends on the choice of λ. Nevertheless, we find that a single λ can usually lead to fairly uniform

performance gains across a large number of benchmarks.

8. Conclusion

In this paper, we have studied the non-conservative off-policy algorithm Peng's Q(λ), and shown that while in the worst case its convergence guarantees are less strong than conservative algorithms such as Retrace, convergence guarantees to the optimal policy are recovered when the behavior policy closely tracks the target policy. This has important consequences for deep RL theory and practice, as this condition often holds when agents are trained through replay buffers, and serves to close the gap between the strong empirical performance observed with non-conservative algorithms in deep RL, and their previous lack of theory.

We expect this to have several important consequences for deep RL theory and practice. Firstly, these results make clear that the *degree* of off-policyness is an important quantity that has real impact on the success of deep RL algorithms, and incorporating quantities related to this into the analysis of off-policy algorithms will be important for developing theoretical understanding of deep RL. Secondly, these findings add weight to growing empirical work highlighting that quantities such as replay buffer size and replay ratio are crucial to the success of deep RL agents (Zhang & Sutton, 2017; Daley & Amato, 2019; Fedus et al., 2020), and deserve further attention.

We believe the analysis presented in this paper is an important step towards a deeper understanding of non-conservative methods, and there are several open questions suitable for future work. For example, the convergence guarantee in Theorem 4 requires $\alpha \geq 1 - \lambda$. However we conjecture that this assumption can be lifted. Besides, while we did not analyze the concentrability coefficients of PQL, Scherrer (2014) reports that conservative policy iteration, which is analogous to PQL, has a better concentrability coefficients. Finally, careful error propagation analyses of gap-increasing algorithms (Azar et al., 2012; Kozuno et al., 2019) and policy-update-regularized algorithms (Vieillard et al., 2020) show a slow update of policies confer the stability against errors on algorithms. In PQL with behavior policy updates, we expect a similar result when α takes an intermediate value.

Acknowledgement

We are grateful to all reviewers for their time spent on reviewing this paper and writing feedback. TK was supported by JSPS KAKENHI Grant Numbers 16H06563. TK thanks Prof. Kenji Doya, Dongqi Han, and Ho Ching Chiu at Okinawa Institute of Science and Technology (OIST) for their valuable comments. TK is also grateful to the research support of OIST to the Neural Computation Unit, where TK partially conducted this research. In particular, TK is thankful for OIST's Scientific Computation and Data Analysis section, which maintains a cluster we used for many of our experiments. YHT acknowledges the computational support from Google Cloud Platform.

References

Achiam, J. Spinning Up in Deep Reinforcement Learning. 2018.

Asadi, K. and Littman, M. L. An Alternative Softmax Operator for Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Azar, M. G., Gómez, V., and Kappen, H. J. Dynamic policy programming. *Journal of Machine Learning Research*, 13(103):3207–3245, 2012.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., and Lillicrap, T. Distributed distributional deterministic policy gradients. In *Proceedings of the International Conference on Learning Representations*, 2018.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bertsekas, D. P. and Ioffe, S. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, Lab. for Info. and Decision Systems Report, MIT, Cambridge, Massachusetts, 1996.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

Casella, G. and Berger, R. L. *Statistical Inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

Coumans, E. and Bai, Y. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

Daley, B. and Amato, C. Reconciling λ-returns with experience replay. In *Advances in Neural Information Processing Systems*, 2019.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. In *Proceedings of the International Conference on Machine Learning*, 2020.

Fox, R., Pakman, A., and Tishby, N. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2016.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning*, 2018.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *Proceedings of the International Conference on Machine Learning*, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 2018.

Harb, J. and Precup, D. Investigating recurrence and eligibility traces in deep Q-networks. *arXiv preprint arXiv:1704.05495*, 2017.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. Q(λ) with off-policy corrections. In *Proceedings of the International Conference on Algorithmic Learning Theory*, 2016.

Hasselt, H. V. Double Q-learning. In *Advances in Neural Information Processing Systems*, 2010.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2002.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015.

Kozuno, T., Uchibe, E., and Doya, K. Theoretical analysis of efficiency and robustness of softmax and gap-increasing operators in reinforcement learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Mousavi, S. S., Schukat, M., Howley, E., and Mannion, P. Applying Q(λ)-learning in deep reinforcement learning to play Atari games. In *AAMAS Workshop on Adaptive Learning Agents*, 2017.

Munos, R. Error bounds for approximate policy iteration. In *Proceedings of the International Conference on Machine Learning*, 2003.

Munos, R. Error bounds for approximate value iteration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2005.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, 2016.

Oh, J., Guo, Y., Singh, S., and Lee, H. Self-imitation learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

Peng, J. and Williams, R. J. Incremental multi-step Q-learning. In *Proceedings of the International Conference on Machine Learning*, 1994.

Peng, J. and Williams, R. J. Incremental multi-step Q-learning. *Machine learning*, 22(1):283–290, March 1996.

Precup, D., Sutton, R. S., and Singh, S. P. Eligibility traces for off-policy policy evaluation. In *Proceedings of the International Conference on Machine Learning*, 2000.

Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994. ISBN 0471619779.

Puterman, M. L. and Shin, M. C. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

Rowland, M., Dabney, W., and Munos, R. Adaptive trade-offs in off-policy learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2020.

Scherrer, B. Performance bounds for λ policy iteration and application to the game of Tetris. *Journal of Machine Learning Research*, 14(1):1181–1227, 2013.

Scherrer, B. Approximate policy iteration schemes: A comparison. In *Proceedings of the International Conference on Machine Learning*, 2014.

Scherrer, B., Gabillon, V., Ghavamzadeh, M., and Geist, M. Approximate modified policy iteration. In *Proceedings of the International Conference on Machine Learning*, 2012.

Scherrer, B., Ghavamzadeh, M., Gabillon, V., Lesner, B., and Geist, M. Approximate modified policy iteration and its application to the game of Tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning*, 2014.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1 edition, 1998.

Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., and Heess, N. dm_control: Software and Tasks for Continuous Control, 2020.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2012.

Vieillard, N., Kozuno, T., Scherrer, B., Pietquin, O., Munos, R., and Geist, M. Leverage the average: an analysis of KL regularization in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. Sample efficient actor-critic with experience replay. In *Proceedings of the International Conference on Learning Representations*, 2017.

Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, Cambridge, UK, May 1989.

Zhang, S. and Sutton, R. S. A deeper look at experience replay. In *NeurIPS Workshop on Deep Reinforcement Learning*, 2017.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2008.