MULTI-STEP GREEDY POLICIES IN MODEL-FREE DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-step greedy policies have been extensively used in model-based Reinforcement Learning (RL) and in the case when a model of the environment is available (e.g., in the game of Go). In this work, we explore the benefits of multi-step greedy policies in model-free RL when employed in the framework of multi-step Dynamic Programming (DP): multi-step Policy and Value Iteration. These algorithms iteratively solve short-horizon decision problems and converge to the optimal solution of the original one. By using model-free algorithms as solvers of the short-horizon problems we derive fully model-free algorithms which are instances of the multi-step DP framework. As model-free algorithms are prone to instabilities w.r.t. the decision problem horizon, this simple approach can help in mitigating these instabilities and results in an improved model-free algorithms. We test this approach and show results on both discrete and continuous control problems.

1 INTRODUCTION

The field of Reinforcement learning (RL) span a wide variety of algorithms for solving decision-making problems through repeated interaction with the environment. By incorporating deep neural networks into RL algorithms, the field of RL has recently witnessed remarkable empirical success (e.g., Mnih et al. 2015; Lillicrap et al. 2015; Levine et al. 2016; Silver et al. 2017). Much of this success had been achieved by model-free RL algorithms, such as Q-learning and policy gradient. These algorithms are known to suffer from high variance in their estimations (Greensmith et al., 2004) and to have difficulties handling function approximation (e.g., Thrun & Schwartz 1993; Baird 1995; Van Hasselt et al. 2016; Lu et al. 2018). These problems are intensified in decision problems with long horizon, i.e., when the discount factor, γ, is large. Although using smaller values of γ addresses the γ-dependent issues and leads to more stable algorithms (Petrik & Scherrer, 2009; Jiang et al., 2015), it comes with a cost, as the algorithm may return a *biased* solution, i.e., it may not converge to an optimal solution of the original decision problem (the one with large value of γ).

Efroni et al. (2018a) recently proposed another approach to mitigate the γ-dependant instabilities in RL in which they study a multi-step greedy versions of the well-known dynamic programming (DP) algorithms policy iteration (PI) and value iteration (VI) (Bertsekas & Tsitsiklis, 1996). Efroni et al. (2018a) also proposed an alternative formulation of the multi-step greedy policy, called κ-greedy policy, and studied the convergence of the resulted PI and VI algorithms: κ-PI and κ-VI. These two algorithms iteratively solve $\gamma\kappa$-discounted decision problems, whose reward has been shaped by the solution of the decision problem at the previous iteration. Unlike the *biased* solution obtained by solving the decision problem with a smaller value of γ, by iteratively solving decision problems with a smaller $\gamma\kappa$ horizon, the κ-PI and κ-VI algorithms could converge to an optimal policy of the original decision problem.

In this work, we derive and empirically validate model-free deep RL (DRL) implementations of κ-PI and κ-VI. In these implementations, we use DQN (Mnih et al., 2015) and TRPO (Schulman et al., 2015) for (approximately) solving $\gamma\kappa$-discounted decision problems (with shaped reward), which is the main component of the κ-PI and κ-VI algorithms. The experiments illustrate the performance of model-free algorithms can be improved by using them as solvers of multi-step greedy PI and VI schemes, as well as emphasize important implementation details while doing so.

2 PRELIMINARIES

In this paper, we assume that the agent's interaction with the environment is modeled as a discrete time γ-discounted Markov Decision Process (MDP), defined by $\mathcal{M}_\gamma = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces; $P \equiv P(s'|s,a)$ is the transition kernel; $R \equiv r(s,a)$

is the reward function with the maximum value of R_{max}; $\gamma \in (0,1)$ is the discount factor; and μ is the initial state distribution. Let $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a stationary Markovian policy, where $\mathcal{P}(\mathcal{A})$ is a probability distribution on the set $\mathcal{A}$. The value of π in any state $s \in \mathcal{S}$ is defined as $V^\pi(s) \equiv \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,\pi(s_t))|s_0 = s,\pi]$, where the expectation is over all the randomness in policy, dynamics, and rewards. Similarly, the action-value function of π is defined as $Q^\pi(s,a) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,\pi(s_t))|s_0 = s, a_0 = a,\pi]$. Since the rewards have the maximum value of R_{max}, both V and Q functions have the maximum value of $V_{\text{max}} = R_{\text{max}}/(1-\gamma)$. An optimal policy π^* is the policy with maximum value at every state. We call the value of π^* the optimal value, and define it as $V^*(s) = \max_\pi \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,\pi(s_t))|s_0 = s,\pi],\ \forall s \in \mathcal{S}$. Furthermore, we denote the state-action value of π^* as $Q^*(s,a)$ and remind the following relation holds $V^*(s) = \max_a Q^*(s,a)$ for all s. The algorithms by which an is be solved (obtain an optimal policy) are mainly based on two popular DP algorithms: Policy Iteration (PI) and Value Iteration (VI). While VI relies on iteratively computing the optimal Bellman operator T applied to the current value function V (Eq. 1), PI relies on (iteratively) calculating a 1-step greedy policy $\pi_{\text{1-step}}$ w.r.t. to the value function of the current policy V (Eq. 2):

$$(TV)(s) = \max_{a\in\mathcal{A}}\ \mathbb{E}[r(s_0,a) + \gamma V(s_1) \mid s_0 = s], \qquad \forall s \in \mathcal{S}, \tag{1}$$

$$\pi_{\text{1-step}}(s) \in \arg\max_{a\in\mathcal{A}}\ \mathbb{E}[r(s_0,a) + \gamma V(s_1) \mid s_0 = s], \quad \forall s \in \mathcal{S}. \tag{2}$$

It is known that T is a γ-contraction w.r.t. the max norm and its unique fixed point is V^*, and the 1-step greedy policy w.r.t. V^* is an optimal policy π^*. In practice, the state space is often large, and thus, we can only approximately compute Eqs. 1 and 2, which results in approximate PI (API) and VI (AVI) algorithms. These approximation errors then propagate through the iterations of the API and AVI algorithms. However, it has been shown that this (propagated) error can be controlled (Munos, 2003; 2005; Farahmand et al., 2010) and after N steps, the algorithms approximately converge to a solution π_N whose difference with the optimal value is bounded (see e.g., Scherrer 2014 for API):

$$\eta(\pi^*) - \eta(\pi_N) \leq C\delta/(1-\gamma)^2 + \gamma^N V_{\max}. \tag{3}$$

In Eq. 3, the scalar $\eta(\pi) = \mathbb{E}_{s\sim\mu}[V^\pi(s)]$ is the expected value function at the initial state,[1] δ represents the per-iteration error, and C upper-bounds the mismatch between the sampling distribution and the distribution according to which the final value function is evaluated (μ in Eq. 3), and depends heavily on the dynamics. Finally, the second term on the RHS of Eq. 3 is the error due to initial values of policy/value, and decays with the number of iterations N.

3 κ-Greedy Policy & κ-PI and κ-VI Algorithms

Algorithm 1 κ-Policy Iteration

Initialize: $\kappa \in [0,1]$, π_0, $N(\kappa)$
for $i = 0,1,\ldots,N(\kappa)-1$ **do**
  $V^{\pi_i} = \mathbb{E}[\sum_{t\geq 0}\gamma^t r_t \mid \pi_i]$
  $\pi_{i+1} \leftarrow \arg\max_\pi\ \mathbb{E}[\sum_{t=0}^{\infty}(\kappa\gamma)^t r_t(\kappa, V^{\pi_i})|\pi]$
end for
Return $\pi_{N(\kappa)}$

Algorithm 2 κ-Value Iteration

Initialize: $\kappa \in [0,1]$, V_0 , $N(\kappa)$
for $i = 0,1,\ldots,N(\kappa)-1$ **do**
  $V_{i+1} = \max_\pi \mathbb{E}[\sum_{t\geq 0}(\gamma\kappa)^t r_t(\kappa, V_i)|\pi]$
end for
$\pi_{N(\kappa)} \leftarrow \arg\max_\pi\ \mathbb{E}[\sum_{t\geq 0}(\kappa\gamma)^t r_t(\kappa, V_{N(\kappa)})|\pi]$
Return $\pi_{N(\kappa)}$

The optimal Bellman operator T (Eq. 1) and 1-step greedy policy $\pi_{\text{1-step}}$ (Eq. 2) can be generalized to multi-step. The most straightforward form of this generalization is by replacing T and $\pi_{\text{1-step}}$ with h-optimal Bellman operator and h-step greedy policy (i.e., a lookahead of horizon h) that are defined by substituting the 1-step return in Eqs. 1 and 2, $r(s_0,a) + \gamma V(s_1)$, with h-step return, $\sum_{t=0}^{h-1} r(s_t,a_t) + \gamma^h V(s_h)$, and computing the maximum over actions $a_0,\ldots,a_{h-1}$, instead of just a_0 (Bertsekas & Tsitsiklis, 1996). Efroni et al. (2018a) proposed an alternative form of multi-step optimal Bellman operator and multi-step greedy policy, called κ-optimal Bellman operator, T_κ, and κ-greedy policy, π_κ, for $\kappa \in [0,1]$, i.e.,

$$(T_\kappa V)(s) = \max_\pi\ \mathbb{E}[\sum_{t\geq 0}(\gamma\kappa)^t r_t(\kappa, V) \mid s_0 = s,\pi], \qquad \forall s \in \mathcal{S}, \tag{4}$$

$$\pi_\kappa(s) \in \arg\max_\pi\ \mathbb{E}[\sum_{t\geq 0}(\gamma\kappa)^t r_t(\kappa, V) \mid s_0 = s,\pi], \quad \forall s \in \mathcal{S}, \tag{5}$$

[1]Note that the LHS of Eq. 3 is the ℓ_1-norm of $(V^{\pi^*} - V^{\pi_N})$ w.r.t. the initial state distribution μ.

where the *shaped reward* $r_t(\kappa, V)$ w.r.t. the value function V is defined as

$$r_t(\kappa, V) \equiv r(s_t, a_t) + (1-\kappa)\gamma V(s_{t+1}). \tag{6}$$

It can be shown that the κ-greedy policy w.r.t. the value function V is the optimal policy w.r.t. a κ-weighted geometric average of all future h-step returns (from $h = 0$ to ∞). This can be interpreted as TD(λ) (Sutton & Barto, 2018) for policy improvement (see Efroni et al., 2018a, Sec. 6). The important difference is that TD(λ) is used for policy evaluation and not for policy improvement.

From Eqs. 4 and 5, it is easy to see that solving these equations is equivalent to solving a surrogate $\gamma\kappa$-discounted MDP with the shaped reward $r_t(\kappa, V)$, which we denote by $\mathcal{M}_{\gamma\kappa}(V)$ throughout the paper. The optimal value of $\mathcal{M}_{\gamma\kappa}(V)$ (the surrogate MDP) is $T_\kappa V$ and its optimal policy is the κ-greedy policy, π_κ. Using the notions of κ-optimal Bellman operator, T_κ, and κ-greedy policy, π_κ, Efroni et al. (2018a) derived κ-PI and κ-VI algorithms, whose pseudocode is shown in Algorithms 1 and 2. κ-PI iteratively *(i)* evaluates the value of the current policy π_i, and *(ii)* set the new policy, π_{i+1}, to the κ-greedy policy w.r.t. the value of the current policy V^{π_i}, by solving Eq. 5. On the other hand, κ-VI repeatedly applies the T_κ operator to the current value function V_i (solves Eq. 4) to obtain the next value function, V_{i+1}, and returns the κ-greedy policy w.r.t. the final value $V_{N(\kappa)}$. Note that for $\kappa = 0$, the κ-greedy policy and κ-optimal Bellman operator are equivalent to their 1-step counterparts, defined by Eqs. 1 and 2, which indicates that κ-PI and κ-VI are generalizations of the seminal PI and VI algorithms.

It has been shown that both PI and VI converge to the optimal value with a linear rate that depends on the discount factor γ, i.e., $\|V^* - V^{\pi_N}\|_\infty \leq O(\gamma^N)$ (see e.g., Bertsekas & Tsitsiklis, 1996; Scherrer, 2013). Analogously, Efroni et al. (2018a) showed that κ-PI and κ-VI converge with faster linear rate of $\xi(\kappa) = \frac{\gamma(1-\kappa)}{1-\gamma\kappa} \leq \gamma$, i.e., $\|V^* - V^{\pi_{N(\kappa)}}\|_\infty \leq O(\xi(\kappa)^{N(\kappa)})$, with the cost that each iteration of these algorithms is computationally more expensive than that of PI and VI. Finally, we state the following two properties of κ-PI and κ-greedy policies that we use in our RL implementations of κ-PI and κ-VI algorithms in Sections 4 and 5:

1) *Asymptotic performance depends on κ.* The following bound that is similar to the one reported in Eq. 3 was proved by Efroni et al. (2018b, Thm. 5) for the performance of κ-PI:

$$\eta(\pi^*) - \eta(\pi_{N(\kappa)}) \leq \underbrace{C(\kappa)\delta(\kappa)/(1-\gamma)^2}_{\text{Asymptotic Term}} + \underbrace{\xi(\kappa)^{N(\kappa)} V_{\max}}_{\text{Decaying Term}}, \tag{7}$$

where $\delta(\kappa)$ and $C(\kappa)$ are quantities similar to δ and C in Eq. 3. Note that the first term on the RHS of Eq. 7 is independent of $N(\kappa)$, while the second one decays with $N(\kappa)$.

2) *Soft updates w.r.t. a κ-greedy policy does not necessarily improve the performance.* Let π_κ be the κ-greedy policy w.r.t. V^π. Then, unlike for 1-step greedy policies, the performance of $(1-\alpha)\pi + \alpha\pi_\kappa$ (soft update) is not necessarily better than that of π (Efroni et al., 2018b, Thm. 1). This hints that it would be advantages to use κ-greedy policies with 'hard' updates (using π_κ as the new policy).

4 RL Implementations of κ-PI and κ-VI

As described in Sec. 3, implementing κ-PI and κ-VI requires iteratively solving a $\gamma\kappa$-discounted surrogate MDP with a shaped reward. If a model of the environment is given, the surrogate MDP can be solved using a DP algorithm (see Efroni et al., 2018a, Sec. 7). When the model is not available, it can be approximately solved by any model-free RL algorithm. In this paper, we focus on the case that the model is not available and propose RL implementations of κ-PI and κ-VI. The main question we investigate in this work is *how model-free RL algorithms should be implemented to efficiently solve the surrogate MDP in κ-PI and κ-VI.*

In this paper, we use DQN (Mnih et al., 2015) and TRPO (Schulman et al., 2015) as subroutines for estimating a κ-greedy policy (Line 4 in κ-PI, Alg. 1 and Line 5 in κ-VI, Alg. 2) or for estimating an optimal value of the surrogate MDP (Line 3 in κ-VI, Alg. 2). For estimating the value of the current policy (Line 3, in κ-PI, Alg. 1), we use standard policy evaluation deep RL (DRL) algorithms.

To implement κ-PI and κ-VI, we shall set the value of $N(\kappa) \in \mathbb{N}$, i.e., the total number of iterations of these algorithms, and determine the number of samples for each iteration. Since $N(\kappa)$ only appears in the second term of Eq. 7, an appropriate choice of $N(\kappa)$ is such that $C(\kappa)\delta(\kappa)/(1-\gamma)^2 \simeq \xi(\kappa)^{N(\kappa)} V_{\max}$. Note that setting $N(\kappa)$ to a higher value would not dramatically improve the

Algorithm 3 κ-PI-DQN

Initialize replay buffer $\mathcal{D}$, Q-networks Q_θ, Q_ϕ, and target networks Q'_θ, Q'_ϕ;
for $i = 0, \ldots, N(\kappa) - 1$ **do**
  # Policy Improvement
  for $t = 1, \ldots, T(\kappa)$ **do**
    Act by an ϵ-greedy policy w.r.t. $Q_\theta(s_t, a)$, observe r_t, s_{t+1}, and store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
    Sample a batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
    Update θ by DQN rule with $\{(s_j, a_j, r_j(\kappa, V_\phi), s_{j+1})\}_{j=1}^N$, where
      $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi_{i-1}(s_{j+1}))$ and $\pi_{i-1}(s) \in \arg\max_a Q'_\theta(s, a)$;
    Copy θ to θ' occasionally $(\theta' \leftarrow \theta)$;
  end for
  # Policy Evaluation of $\pi_i(s) \in \arg\max_a Q'_\theta(s, a)$
  for $t = 1, \ldots, T(\kappa)$ **do**
    Sample a batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
    Update ϕ by TD(0) off-policy rule with $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$, and $\pi_i(s) \in \arg\max_a Q'_\theta(s, a)$;
    Copy ϕ to ϕ' occasionally $(\phi' \leftarrow \phi)$;
  end for
end for

performance, because the asymptotic term in Eq. 7 is independent of $N(\kappa)$. In practice, since $\delta(\kappa)$ and $C(\kappa)$ are unknown, we set $N(\kappa)$ to satisfy the following equality:

$$\xi(\kappa)^{N(\kappa)} = C_{FA}, \tag{8}$$

where C_{FA} is a hyper-parameter that depends on the *final-accuracy* we are aiming for. For example, if we expect the final accuracy being 90%, we would set $C_{FA} = 0.1$. Our results suggest that this approach leads to a reasonable choice for $N(\kappa)$, e.g., $N(\kappa = 0.99) \simeq 4$ and $N(\kappa = 0.5) \simeq 115$, for $C_{FA} = 0.1$ and $\gamma = 0.99$. As we increase κ, we expect less iterations are needed for κ-PI and κ-VI to converge to a good policy. Another important observation is that since the discount factor of the surrogate MDP that κ-PI and κ-VI solve at each iteration is $\gamma\kappa$, the *effective horizon* (the effective horizon of a $\gamma\kappa$-discounted MDP is $1/(1 - \gamma\kappa)$) of the surrogate MDP increases with κ.

Lastly, we need to determine the number of samples for each iteration of κ-PI and κ-VI. We allocate equal number of samples per iteration, denoted by $T(\kappa)$. Since the total number of samples, T, is known beforehand, we set the number of samples per iteration to

$$T(\kappa) = T/N(\kappa). \tag{9}$$

5 DQN AND TRPO IMPLEMENTATIONS OF κ-PI AND κ-VI

In this section, we study the use of DQN (Mnih et al., 2015) and TRPO (Schulman et al., 2015) in κ-PI and κ-VI algorithms. We first derive our DQN and TRPO implementations of κ-PI and κ-VI in Sections 5.1 and 5.2. We refer to the resulting algorithms as κ-PI-DQN, κ-VI-DQN, κ-PI-TRPO, and κ-VI-TRPO. It is important to note that for $\kappa = 1$, κ-PI-DQN and κ-VI-DQN are reduced to DQN, and κ-PI-TRPO and κ-VI-TRPO are reduced to TRPO. We then conduct a set of experiments with these algorithms, in Sections 5.1.1 and 5.2.1, in which we carefully study the effect of κ and $N(\kappa)$ (or equivalently the hyper-parameter C_{FA}, defined by Eq. 8) on their performance. In these experiments, we specifically focus on answering the following questions:

1. Is the performance of DQN and TRPO improve when using them as κ-greedy solvers in κ-PI and κ-VI? Is there a performance tradeoff w.r.t. to κ?
2. Following κ-PI and κ-VI, our DQN and TRPO implementations of these algorithms devote a significant number of sample $T(\kappa)$ to each iteration. Is this needed or a 'naive' choice of $T(\kappa) = 1$, or equivalently $N(\kappa) = T$, works just well, for all values of κ?

5.1 DQN IMPLEMENTATION OF κ-PI AND κ-VI

Algorithm 3 contains the pseudo-code of κ-PI-DQN. Due to space constraints, we report its detailed pseudo-code in Appendix A.1 (Alg. 5). In the *policy improvement stage* of κ-PI-DQN, we use DQN to solve the $\gamma\kappa$-discounted surrogate MDP with the shaped reward $r_t(\kappa, V_\phi \simeq V^{\pi_{i-1}})$,

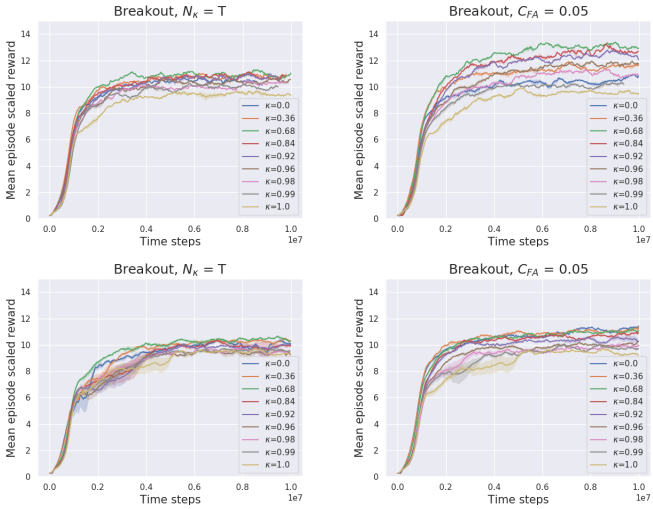

Figure 1: Training performance of κ-PI-DQN (Top) and κ-VI-DQN (Bottom) on Breakout, for the hyper-parameter $C_{FA} = 0.05$ (right) and for the 'naive' baseline $N(\kappa) = T$ (left).

i.e., at the end of this stage $\mathcal{M}_{\gamma\kappa}(V_\phi)$. The output of the DQN is approximately the optimal Q-function of $\mathcal{M}_{\gamma\kappa}(V_\phi)$, and thus, the κ-greedy policy w.r.t. V_ϕ is equal to $\arg\max_a Q_\theta(\cdot, a)$. At the *policy evaluation stage*, we use off-policy TD(0) to evaluate the Q-function of the current policy π_i, i.e., $Q_\phi \simeq Q^{\pi_i}$. Although what is needed on Line 8 is an estimate of the value function of the current policy, $V_\phi \simeq V^{\pi_{i-1}}$, we chose to evaluate the Q-function of π_i: the data in our disposal (the transitions stored in the replay buffer) is an off-policy data and the Q-function of a fixed policy can be easily evaluated with this type of a data using off-policy TD(0), unlike the value function.

Remark 1 *In order for V_ϕ to be an accurate estimate of the value function of π_{i-1} on Line 8, we should use an additional target network, $\widetilde{Q}'_\theta$, that remains unchanged during the policy improvement stage. This network should be used in $\pi_{i-1}(\cdot) = \arg\max_a \widetilde{Q}'_\theta(\cdot, a)$ on Line 8, and be only updated right after the improvement stage on Line 11. However, to reduce the space complexity of the algorithm, we do not use this additional target network and compute π_{i-1} on Line 8 as $\arg\max Q'_\theta$, despite the fact that Q'_θ changes during the improvement stage.*

We report the pseudo-code of κ-VI-DQN in Appendix A.1 (Alg. 6). Note that κ-VI simply repeats $V \leftarrow T_\kappa V$ and computes $T_\kappa V$, which is the optimal value of the surrogate MDP $\mathcal{M}_{\gamma\kappa}(V)$. In κ-VI-DQN, we repeatedly solve $\mathcal{M}_{\gamma\kappa}(V)$ by DQN, and use its optimal Q-function to shape the reward of the next iteration. Let $Q^*_{\gamma\kappa,V}$ and $V^*_{\gamma\kappa,V}$ be the optimal Q and V functions of $\mathcal{M}_{\gamma\kappa}(V)$. Then, we have $\max_a Q^*_{\gamma\kappa,V}(s,a) = V^*_{\gamma\kappa,V}(s) = (T_\kappa V)(s)$, where the first equality is by definition (Sec. 2) and the second one holds since $T_\kappa V$ is the optimal value of $\mathcal{M}_{\gamma\kappa}(V)$ (Sec. 3). Therefore, in κ-VI-DQN, we shape the reward of each iteration by $\max_a Q_\phi(s,a)$, where Q_ϕ is the output of the DQN from the previous iteration, i.e., $\max_a Q_\phi(s,a) \simeq T_\kappa V_{i-1}$.

5.1.1 κ-PI-DQN AND κ-VI-DQN EXPERIMENTS

In this section, we empirically analyze the performance of the κ-PI-DQN and κ-VI-DQN algorithms on the Atari domains: Breakout, Seaquest, SpaceInvaders, and Enduro (Bellemare et al., 2013). We start by performing an ablation test on three values of parameter $C_{FA} = \{0.001, 0.05, 0.2\}$ on the Breakout domain. The value of C_{FA} sets the number of samples per iteration $T(\kappa)$ (Eq. 8) and the total number of iterations $N(\kappa)$ (Eq. 9). Aside from C_{FA}, we set the total number of samples to $T \simeq 10^6$. This value represents the number of samples after which our DQN-based algorithms approximately converge. For each value of C_{FA}, we test κ-PI-DQN and κ-VI-DQN for several κ values. In both algorithms, the best performance was obtained with $C_{FA} = 0.05$, thus, we set $C_{FA} = 0.05$ in our experiments with other Atari domains.

Domain	Alg.	κ_{best}	$\kappa = 0$	DQN, $\kappa = 1$	$N(\kappa) = T$, κ_{best}
Breakout	κ-PI	224($\pm$5), κ=0.68	160($\pm$3)	131($\pm$3)	171($\pm$1), λ=0.68
	κ-VI	180($\pm$5), κ=0.68	179($\pm$6)		
SpaceInv.	κ-PI	747($\pm$23), κ=0.84	611($\pm$15)	685($\pm$24)	695($\pm$16), λ=0.92
	κ-VI	707($\pm$32), κ=0.36	669($\pm$16)		
Seaquest	κ-PI	5159($\pm$509), κ=0.84	2732($\pm$281)	3207($\pm$248)	4371($\pm$466), λ=0.84
	κ-VI	3394($\pm$86), κ=0.36	2631($\pm$496)		
Enduro	κ-PI	544($\pm$29), κ=0.84	371($\pm$215)	355($\pm$52)	547($\pm$17), λ=0.68
	κ-VI	499($\pm$18), κ=0.84	492 ($\pm$28)		

Table 1: The final training performance of κ-PI-DQN and κ-VI-DQN on the Atari domains, for the hyper-parameter $C_{FA} = 0.05$.

Figure 1 shows the training performance of κ-PI-DQN (Top) and κ-VI-DQN (Bottom) for the best value of $C_{FA} = 0.05$, as well as for the 'naive' baseline $T(\kappa) = 1$, or equivalently $N(\kappa) = T$, on Breakout. The results on Breakout for the other values of C_{FA} and the results on the other Atari domains for $C_{FA} = 0.05$ have been reported in Appendix A.2. Table 1 shows the final training performance of κ-PI-DQN and κ-VI-DQN on the Atari domains with $C_{FA} = 0.05$. Note that the scores reported in Table 1 are the actual returns of the Atari domains, while the vertical axis in the plots of Figure 1 corresponds to a scaled return. We plot the scaled return, since this way it would be easier to reproduce our results using the OpenAI Baselines codebase (Hill et al., 2018).

The results of Fig. 1 and Table 1, as well as those in Appendix A.2, exhibit that both κ-PI-DQN and κ-VI-DQN improve the performance of DQN ($\kappa = 1$). Moreover, they show that setting $N(\kappa) = T$ leads to a clear degradation of the final training performance on all of the domains expect Enduro, which attains better performance for $N(\kappa) = T$. Although the performance degrades, the results for $N(\kappa) = T$ are still better than for DQN.

5.2 TRPO Implementation of κ-PI and κ-VI

Algorithm 4 contains the pseudo-code of κ-PI-TRPO (detailed pseudo-code in Appendix A.1). TRPO iteratively updates the current policy using its return and an estimate of its value function. In our κ-PI-TRPO, at each iteration i: **1)** we use the estimate of the current policy $V'_\phi \simeq V^{\pi_{i-1}}$ (computed in the previous iteration) to calculate the return $R(\kappa, V'_\phi)$ and an estimate of the value function V_θ of the surrogate MDP $\mathcal{M}_{\gamma\kappa}(V'_\phi)$, **2)** we use the return $R(\kappa, V'_\phi)$ and V_θ to compute the new policy π_i, and **3)** we estimate the value of the new policy $V_\phi \simeq V^{\pi_i}$ on the original, γ discounted, MDP.

In Appendix B.1 we provide the pseudocode of κ-VI-TRPO derived by the κ-VI meta algorithm. As previously noted, κ-VI iteratively solves the $\gamma\kappa$ discounted surrogate MDP and uses its optimal value $T_\kappa V_{i-1}$ to shape the reward of the surrogated MDP in the i'th iteration. With that in mind, consider κ-PI-TRPO. Notice that as π_θ converges to the optimal policy of the surrogate $\gamma\kappa$ discounted MDP, $V_{\bar{\theta}}$ converges to the optimal value of the surrogate MDP, i.e., it converges to $T_\kappa V_{i-1} = T_\kappa V_{i-1}$. Thus, κ-PI-TRPO can be turn to κ-VI-TRPO by eliminating the policy evaluation stage, and simply copy $\phi \leftarrow \bar{\theta}$, meaning, $V_\phi \leftarrow V_{\bar{\theta}} = T_\kappa V_\phi$.

5.2.1 κ-PI-TRPO and κ-VI-TRPO Experiments

In this section, we empirically analyze the performance of the κ-PI-TRPO and κ-VI-TRPO algorithms on the MuJoCo domains: Walker2d-v2, Ant-v2, HalfCheetah-v2, HumanoidStandup-v2, and Swimmer-v2, (Todorov et al., 2012). As in Section 5.1.1, we start by performing an ablation test on the parameter $C_{FA} = \{0.001, 0.05, 0.2\}$ on the Walker domain. We set the total number of iterations to 2000, with each iteration consisting 1000 samples. Thus, the total number of samples is $T \simeq 2 \times 10^6$. This is the number of samples after which our TRPO-based algorithms approximately converge. For each value of C_{FA}, we test κ-PI-TRPO and κ-VI-TRPO for several κ values. In both algorithms, the best performance was obtained with $C_{FA} = 0.2$, thus, we set $C_{FA} = 0.2$ in our experiments with other MuJoCo domains.

Figure 2 shows the training performance of κ-PI-TRPO (Top) and κ-VI-TRPO (Bottom) for the best value of $C_{FA} = 0.2$, as well as for the 'naive' baseline $T(\kappa) = 1$, or equivalently $N(\kappa) = T$, on Walker. The results on Walker for the other C_{FA} values and the other MuJoCo domains for

Algorithm 4 κ-PI-TRPO

Initialize V-networks V_θ and V_ϕ, policy network π_ψ, and target network V'_ϕ;
for $i = 0, \ldots, N(\kappa) - 1$ **do**
  for $t = 1, \ldots, T(\kappa)$ **do**
    Simulate the current policy π_ψ for M steps and calculate the following two returns for all steps j:
      $R_j(\kappa, V'_\phi) = \sum_{t=j}^{M} (\gamma\kappa)^{t-j} r_t(\kappa, V'_\phi)$ and $\rho_j = \sum_{t=j}^{M} \gamma^{t-j} r_t$;
    Update θ by minimizing the batch loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V'_\phi))^2$;
    # Policy Improvement
    Update ψ using TRPO by the batch $\{(R_j(\kappa, V'_\phi), V_\theta(s_j))\}_{j=1}^{N}$;
    # Policy Evaluation
    Update ϕ by minimizing the batch loss function: $\mathcal{L}_{V_\phi} = \frac{1}{N} \sum_{j=1}^{N} (V_\phi(s_j) - \rho_j)^2$;
  end for
  Copy ϕ to ϕ' $(\phi' \leftarrow \phi)$;
end for

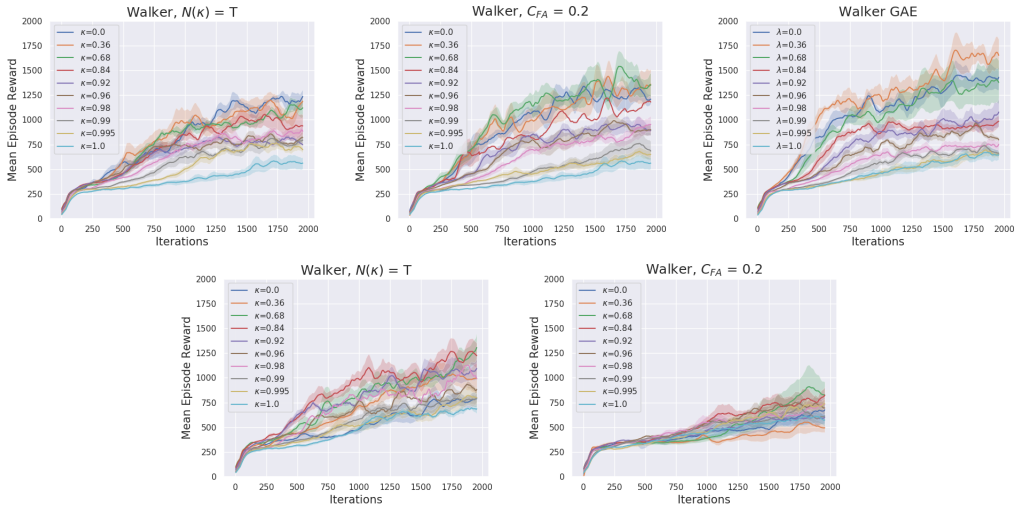

Figure 2: Training performance of κ-PI-TRPO (Top) and κ-VI-TRPO (Bottom) on Walker, for the hyper-parameter $C_{FA} = 0.2$ (right) and for the 'naive' baseline $N(\kappa) = T$ (left).

$C_{FA} = 0.2$ have been reported in Appendix B.3. Table 2 shows the final training performance of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains with $C_{FA} = 0.2$.

The results of Figure 2 and Table 2, as well as those in Appendix B.3, exhibit that both κ-PI-TRPO and κ-VI-TRPO yield better performance than TRPO ($\kappa = 1$). Furthermore, they show that the algorithms with $C_{FA} = 0.2$ perform better than with $N(\kappa) = T$. However, the improvement is less significant relative to the DQN-based results in Section 5.1.1.

5.2.2 Comparison with the Generalized Advantage Estimation algorithm

There is an intimate relation between κ-PI and the GAE algorithm Schulman et al. (2016) which we elaborate on in this section. In GAE the policy is updated by the gradient:

$$\nabla_\theta \mathbb{E}_{s\sim\mu}[V^{\pi_\theta}(s)] = \mathbb{E}_{s_0\sim d_{\mu,\pi}} \left[\nabla_\theta \log \pi_\theta(s_0) \sum_t (\gamma\lambda)^t \delta(V) \right]; \ \delta(V) = r_t + \gamma V_{t+1} - V_t, \quad (10)$$

which can be interpreted as a gradient step in a $\gamma\lambda$ discounted MDP with rewards $\delta(V)$, which we refer here as $\mathcal{M}_{\gamma\lambda}^{\delta(V)}$. As noted in Efroni et al. (2018a), Section 6, the optimal policy of the MDP $\mathcal{M}_{\gamma\lambda}^{\delta(V)}$ is the optimal policy of $\mathcal{M}_{\gamma\kappa}(V)$ with $\kappa = \lambda$, i.e., the κ-greedy policy w.r.t. V: thus, the optimal policy of $\mathcal{M}_{\gamma\lambda}^{\delta(V)}$ is the κ-greedy policy w.r.t. V. GAE, instead of solving the κ-greedy policy while keeping V fixed, changes the policy and updates V by the return concurrently. Thus,

Domain	Alg.	κ_{best}	$\kappa = 0$	TRPO, $\kappa = 1$	$N(\kappa) = T$, κ_{best}	GAE
Walker	κ-PI κ-VI	1352($\pm$233), κ=0.68 827($\pm$269), κ=0.68	1205 ($\pm$99) 669($\pm$125)	560 ($\pm$117)	1158($\pm$75), κ=0.36	1664($\pm$318), λ=0.36
Ant	κ-PI κ-VI	1359($\pm$326), κ=0.68 2916($\pm$455), κ=0.68	1083($\pm$205) 1809($\pm$342)	-18.47($\pm$2)	1225($\pm$141), κ=0.0	1152($\pm$255), λ=0.0
HalfCheetah	κ-PI κ-VI	1367($\pm$406), κ=0.36 1735($\pm$800), κ=0.36	855($\pm$160) 1078($\pm$48)	74($\pm$202)	1450($\pm$200), κ=0.36	1453($\pm$203), λ=0.36
HumanoidStand	κ-PI κ-VI	73743($\pm$1988), κ=0.99 74063($\pm$1779), κ=0.99	73486($\pm$1211) 51323($\pm$1805)	67545($\pm$1545)	72588($\pm$1929), κ=0.98	71420($\pm$1401), λ=0.98
Swimmer	κ-PI κ-VI	108($\pm$17), κ=1.0 108($\pm$17), κ=1.0	43($\pm$3) 46($\pm$1)	108($\pm$17)	108($\pm$17), κ=1.0	108($\pm$14), $\lambda = 1.0$

Table 2: The final training performance of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains, for the hyper-parameter $C_{FA} = 0.2$.

Figure 3: Lowering the discount factor

this approach is conceptually similar to κ-PI-TRPO with $N(\kappa) = T$. There, the value and policy are concurrently updated as well, without clear separation between the update of the policy and the value.

In Figure 2 and Table 2 the performance of GAE is compared to the one of κ-PI-TRPO and κ-VI-TRPO. The performance of the latter two is slightly better than the one of GAE.

Remark 2 (Implementation of GAE) *We used the OpenAI baseline implementation of GAE with a small modification. In the baseline code, the value network is updated w.r.t. to the target $\sum_t (\gamma\lambda)^t r_t$, whereas in Schulman et al. (2016) the authors used the target $\sum_t \gamma^t r_t$ (see Schulman et al. (2016), Eq.28). We chose the latter form in our implementation to be in accord with Schulman et al. (2016).*

5.3 DQN and TRPO Performance Versus the Discount Factor

To supply with a more complete view on our experiments, we tested the performance of the "vanilla" DQN and TRPO when trained with different γ values than the previously used one ($\gamma = 0.99$). As evident in Figure 3, only for the Ant domain this approach resulted in improved performance when for TRPO trained with $\gamma = 0.68$. It is interesting to observe that for the Ant domain the performance of κ-PI-TRPO and especially of κ-VI-TRPO (Table 2) significantly surpassed the one of TRPO trained with $\gamma = 0.68$. The performance of DQN and TRPO on the Breakout, SpaceInvaders and Walker domains decreased or remained unchanged in the tested γ values. Thus, on these domains, changing the discount factor does not improve the DQN and TRPO algorithms, as using κ-PI or κ-VI with smaller κ value do.

It is interesting to observe that the performance on the Mojoco domains for small γ, e.g., $\gamma = 0.68$, achieved good performance, whereas for the Atari domains the performance degraded with lowering γ. This fits the nature of these domains: in the Mojoco domains the decision problem inherently has much shorter horizon than in the Atari domains.

Furthermore, it is important to stress that γ and κ are two different parameters an algorithm designer may use. For example, one can perform a scan of γ value, fix γ to the one with optimal performance, and then test the performance of different κ values.

6 Conclusion and Future Work

In this work we formulated and empirically tested generalizations of DQN and TRPO derived by the theory of multi-step DP and, specifically, of κ-PI and κ-VI algorithms. In the vast majority of our experiments the derived algorithms improved over the performance of their "vanilla" counterparts (i.e., $\kappa = 1$), except for the Swimmer domain from the Mojoco suite. Importantly, the performance of the algorithms was shown to be 'smooth' in the parameter κ. This suggests careful hyperparameter tuning of κ is not of great necessity. Furthermore, our experiments demonstrated the usefulness of the theoretical framework, especially the importance of choosing $N(\kappa)$ (Eq. 8) in a proper way.

The framework of κ-PI and κ-VI resulted in simple generalizations of DQN and TRPO. The main reason for this simplicity lies in the fact that our algorithms are based on solving discounted MDPs. Indeed, an interesting future work would be to formulate and test model-free algorithms which solve a lookahead policy, or an h-greedy policy (Bertsekas & Tsitsiklis, 1996; Efroni et al., 2018a), as oppose to the κ-greedy policy used in this work. Lastly, an important future work in continuation to our work is to study algorithms which change κ in an adaptive manner. This, we believe, would greatly improve the resulting methods.

References

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37, 1995.

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

D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*, volume 5. 1996.

Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. Beyond the one step greedy approach in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, 2018a.

Y. Efroni, G. Dalal, B. Scherrer, and S. Mannor. Multiple-step greedy policies in approximate and online reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5238–5247, 2018b.

A. Farahmand, C. Szepesvári, and R. Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, pp. 568–576, 2010.

E. Greensmith, P. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004.

A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

N. Jiang, A. Kulesza, S. Singh, and R. Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1181–1189, 2015.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

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

T. Lu, D. Schuurmans, and C. Boutilier. Non-delusional q-learning and value iteration. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 9971–9981, 2018.

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

R. Munos. Error bounds for approximate policy iteration. In *Proceedings of the 20th International Conference on Machine Learning*, pp. 560–567, 2003.

R. Munos. Error bounds for approximate value iteration. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pp. 1006–1011, 2005.

M. Petrik and B. Scherrer. Biasing approximate dynamic programming with a lower discount factor. In *Advances in neural information processing systems*, pp. 1265–1272, 2009.

B. Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. In *Advances in Neural Information Processing Systems*, pp. 386–394, 2013.

B. Scherrer. Approximate policy iteration schemes: a comparison. In *International Conference on Machine Learning*, pp. 1314–1322, 2014.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust-region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550 (7676):354, 2017.

R. Sutton and A. Barto. *Reinforcement learning: An introduction*. 2018.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Connectionist Models Summer School*, pp. 255–263, 1993.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

A DQN IMPLEMENTATION OF κ-PI AND κ-VI

A.1 PSEUDO-CODES

In this section, we report the detailed pseudo-codes of the κ-PI-DQN and κ-VI-DQN algorithms, described in Section 5.1, side-by-side.

Algorithm 5 κ-PI-DQN

Initialize replay buffer $\mathcal{D}$, and Q-networks Q_θ and Q_ϕ with random weights θ and ϕ;
Initialize target networks Q'_θ and Q'_ϕ with weights $\theta' \leftarrow \theta$ and $\phi' \leftarrow \phi$;
for $i = 0, \dots, N(\kappa) - 1$ **do**
  # Policy Improvement
  for $t = 1, \dots, T(\kappa)$ **do**
    Select a_t as an ϵ-greedy action w.r.t. $Q_\theta(s_t, a)$;
    Execute a_t, observe r_t and s_{t+1}, and store the tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
    Update θ by minimizing the following loss function:
      $\mathcal{L}_{\mathcal{Q}_\theta} = \frac{1}{N}\sum_{j=1}^N \big(Q_\theta(s_j, a_j) - (r_j(\kappa, V_\phi) + \gamma\kappa \max_a Q'_\theta(s_{j+1}, a))\big)^2$, where
      $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi_{i-1}(s_{j+1}))$ and $\pi_{i-1}(s_{j+1}) \in \arg\max_a Q'_\theta(s_{j+1}, a)$;
    Copy θ to θ' occasionally $(\theta' \leftarrow \theta)$;
  end for
  # Policy Evaluation
  Set $\pi_i(s) \in \arg\max_a Q'_\theta(s, a)$;
  for $t' = 1, \dots, T(\kappa)$ **do**
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
    Update ϕ by minimizing the following loss function:
      $\mathcal{L}_{\mathcal{Q}_\phi} = \frac{1}{N}\sum_{j=1}^N \big(Q_\phi(s_j, a_j) - (r_j + \gamma Q'_\phi(s_{j+1}, \pi_i(s_{j+1})))\big)^2$;
    Copy ϕ to ϕ' occasionally $(\phi' \leftarrow \phi)$;
  end for
end for

Algorithm 6 κ-VI-DQN

Initialize replay buffer $\mathcal{D}$, and Q-networks Q_θ and Q_ϕ with random weights θ and ϕ;
Initialize target network Q'_θ with weights $\theta' \leftarrow \theta$;
for $i = 0, \dots, N(\kappa) - 1$ **do**
  # Evaluate $T_\kappa V_\phi$ and the κ-greedy policy w.r.t. V_ϕ
  for $t = 1, \dots, T(\kappa)$ **do**
    Select a_t as an ϵ-greedy action w.r.t. $Q_\theta(s_t, a)$;
    Execute a_t, observe r_t and s_{t+1}, and store the tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
    Update θ by minimizing the following loss function:
      $\mathcal{L}_{\mathcal{Q}_\theta} = \frac{1}{N}\sum_{j=1}^N \big(Q_\theta(s_j, a_j) - (r_j(\kappa, V_\phi) + \kappa\gamma \max_a Q'_\theta(s_{j+1}, a))\big)^2$, where
      $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi(s_{j+1}))$ and $\pi(s_{j+1}) \in \arg\max_a Q'_\theta(s_{j+1}, a)$;
    Copy θ to θ' occasionally $(\theta' \leftarrow \theta)$;
  end for
  Copy θ to ϕ $(\phi \leftarrow \theta)$
end for

Hyperparameter	Value
Horizon (T)	1000
Adam stepsize	1×10^{-4}
Target network update frequency	1000
Replay memory size	100000
Discount factor	0.99
Total training time steps	10000000
Minibatch size	32
Initial exploration	1
Final exploration	0.1
Final exploration frame	1000000
#Runs used for plot averages	4
Confidence interval for plot runs	50%

Table 3: Hyperparameters for κ-PI-DQN and κ-VI-DQN.

A.2 Ablation Test for C_{FA}

In this section, we report additional results of the application of κ-PI-DQN and κ-VI-DQN on the Atari domains. A summary of these results has been reported in Table 1 in the main paper.

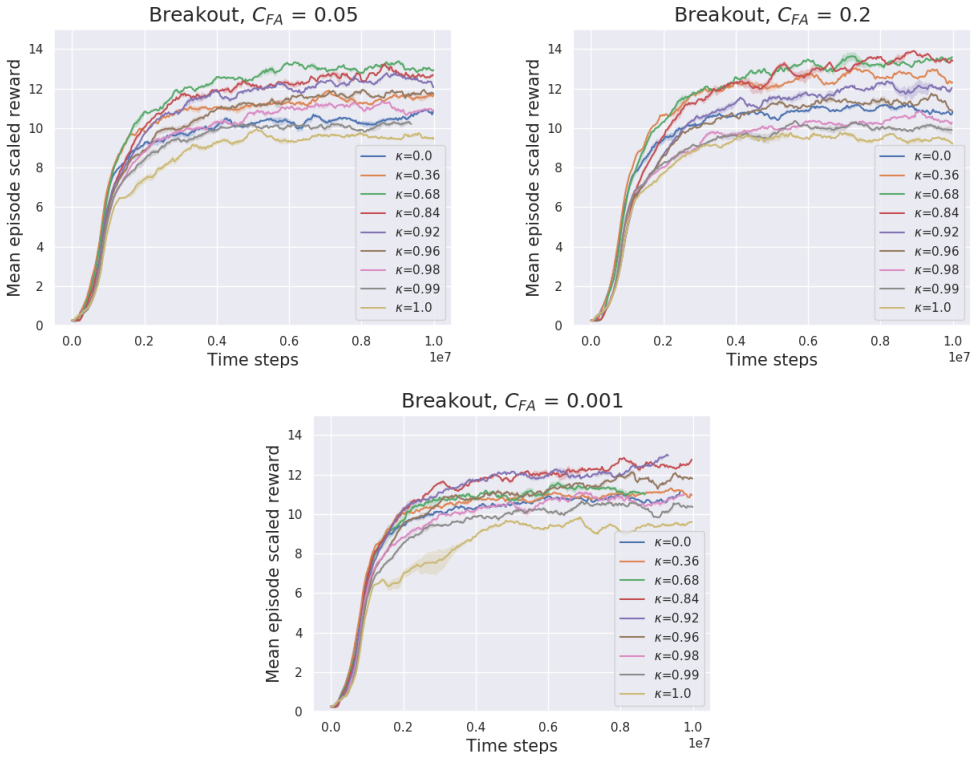

Figure 4: Performance of κ-PI-DQN and κ-VI-DQN on Breakout for different values of C_{FA}.

A.3 κ-PI-DQN AND κ-VI-DQN PLOTS

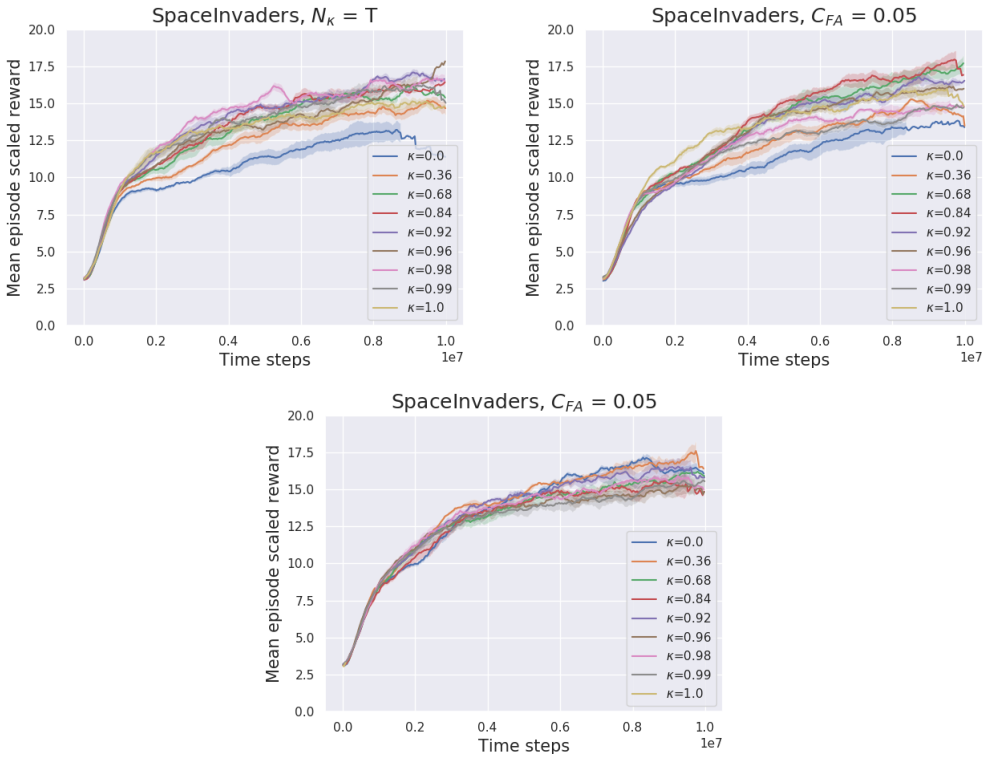

Figure 5: Training performance of κ-PI-DQN (Top) and κ-VI-DQN (Bottom) on SpaceInvaders, for the hyper-parameter $C_{FA} = 0.05$ (right) and for the 'naive' baseline $N(\kappa) = T$ (left).

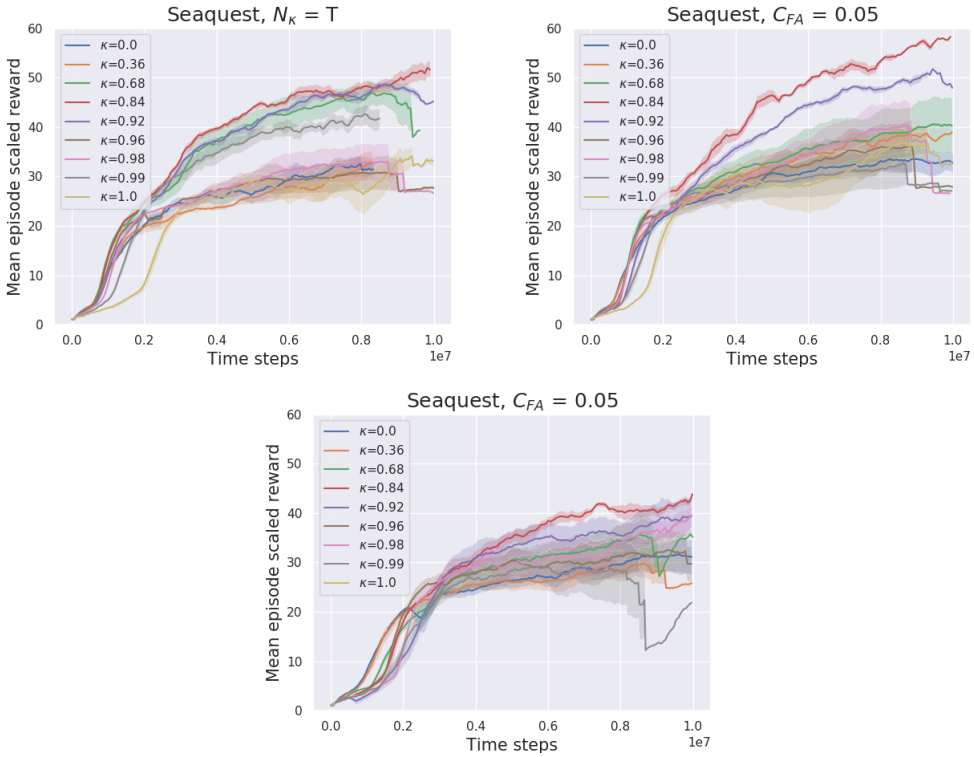

Figure 6: Training performance of κ-PI-DQN (Top) and κ-VI-DQN (Bottom) on Seaquest, for the hyper-parameter $C_{FA} = 0.05$ (right) and for the 'naive' baseline $N(\kappa) = T$ (left).

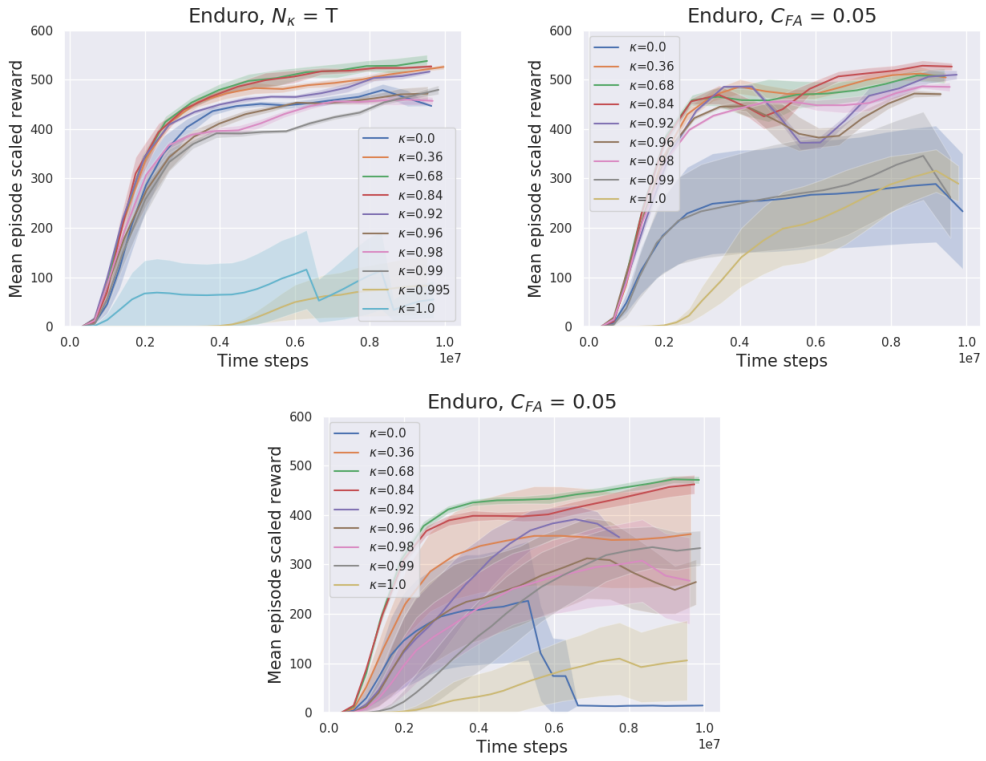

Figure 7: Training performance of κ-PI-DQN (Top) and κ-VI-DQN (Bottom) on Enduro, for the hyper-parameter $C_{FA} = 0.05$ (right) and for the 'naive' baseline $N(\kappa) = T$ (left).

B TRPO IMPLEMENTATION OF κ-PI AND κ-VI

B.1 PSEUDO-CODES

In this section, we report the detailed pseudo-codes of the κ-PI-TRPO and κ-VI-TRPO algorithms, described in Section 5.2, side-by-side.

Algorithm 7 κ-PI-TRPO

Initialize V-networks V_θ and V_ϕ, and policy network π_ψ with random weights θ, ϕ, and ψ
Initialize target network V'_ϕ with weights $\phi' \leftarrow \phi$
for $i = 0, \ldots, N(\kappa) - 1$ **do**
  for $t = 1, \ldots, T(\kappa)$ **do**
    Simulate the current policy π_ψ for M time-steps;
    for $j = 1, \ldots, M$ **do**
      Calculate $R_j(\kappa, V'_\phi) = \sum_{t=j}^{M} (\gamma\kappa)^{t-j} r_t(\kappa, V'_\phi)$ and $\rho_j = \sum_{t=j}^{M} \gamma^{t-j} r_t$;
    end for
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update θ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V'_\phi))^2$;
    # Policy Improvement
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update ψ using TRPO with advantage function computed by $\{(R_j(\kappa, V'_\phi), V_\theta(s_j))\}_{j=1}^{N}$;
    # Policy Evaluation
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update ϕ by minimizing the loss function: $\mathcal{L}_{V_\phi} = \frac{1}{N} \sum_{j=1}^{N} (V_\phi(s_j) - \rho_j)^2$;
  end for
  Copy ϕ to ϕ' $(\phi' \leftarrow \phi)$;
end for

Algorithm 8 κ-VI-TRPO

Initialize V-network V_θ and policy network π_ψ with random weights θ and ψ;
Initialize target network V'_ϕ with weights $\phi'; \leftarrow \theta$;
for $i = 0, \ldots, N(\kappa) - 1$ **do**
  # Evaluate $T_\kappa V'_\phi$ and the κ-greedy policy w.r.t. V'_ϕ
  for $t = 1, \ldots, T(\kappa)$ **do**
    Simulate the current policy π_ψ for M time-steps;
    for $j = 1, \ldots, M$ **do**
      Calculate $R_j(\kappa, V'_\phi) = \sum_{t=j}^{M} (\gamma\kappa)^{t-j} r_t(\kappa, V'_\phi)$;
    end for
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps
    Update θ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V'_\phi))^2$;
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps
    Update ψ using TRPO with advantage function computed by $\{(R_j(\kappa, V'_\phi), V_\theta(s_j))\}_{j=1}^{N}$;
  end for
  Copy θ to ϕ' $(\phi' \leftarrow \theta)$;
end for

B.2 ABLATION TEST FOR C_{FA}

In this section, we report additional results of the application of κ-PI-TRPO and κ-VI-TRPO on the Atari domains. A summary of these results has been reported in Table 2 in the main paper.

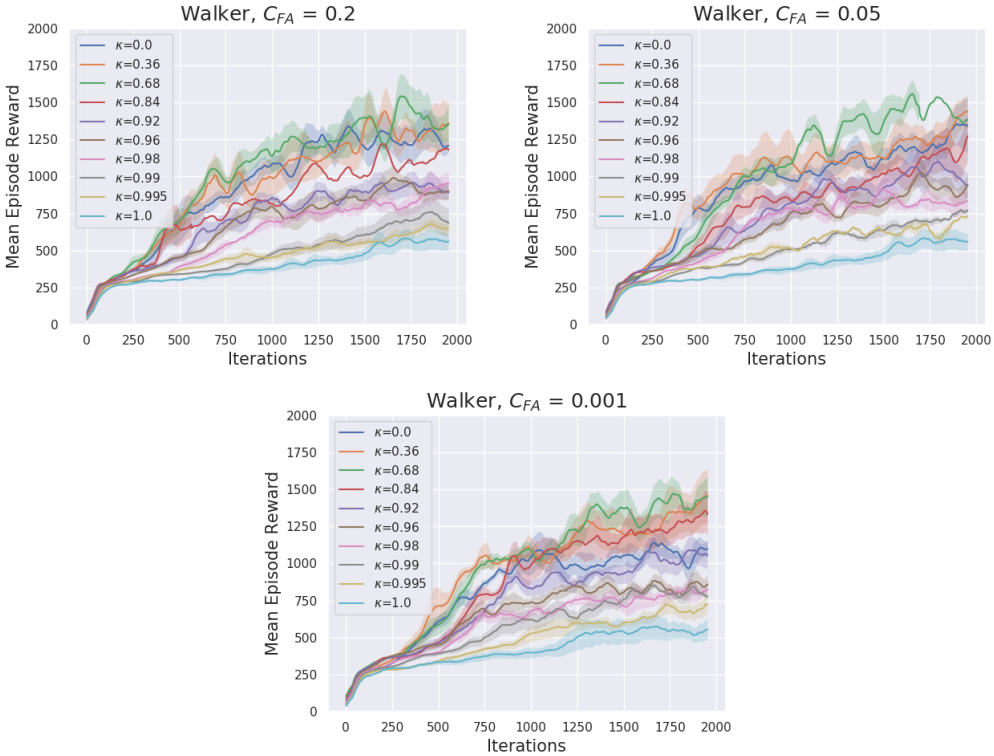

Figure 8: Performance of κ-PI-TRPO and κ-VI-TRPO on Walker2d-v2 for different values of C_{FA}.

Hyperparameter	Value
Horizon (T)	1000
Adam stepsize	1×10^{-3}
Number of samples per Iteration	1024
Entropy coefficient	0.01
Discount factor	0.99
Number of Iterations	2000
Minibatch size	128
#Runs used for plot averages	5
Confidence interval for plot runs	50%

Table 4: Hyper-parameters of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains.

B.3 κ-PI-TRPO AND κ-VI-TRPO PLOTS

In this section, we report additional results of the application of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains. A summary of these results has been reported in Table 2 in the main paper.

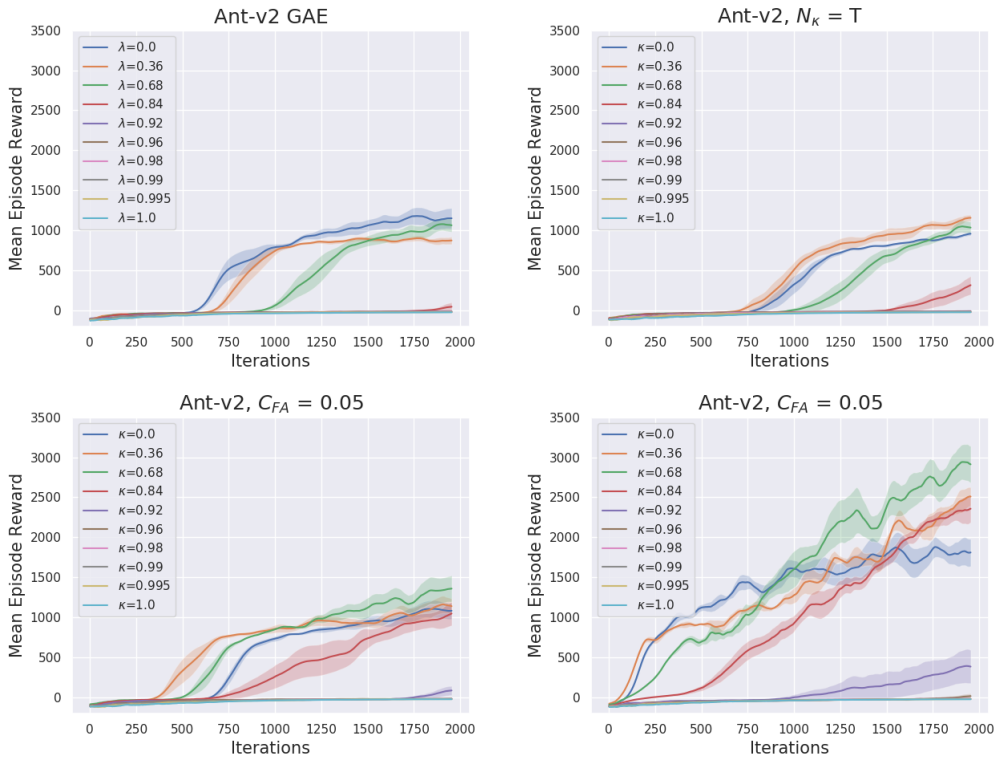

Figure 9: Performance of GAE and 'Naive' baseline (Top row) and κ-PI-TRPO (Bottom left) and κ-VI-TRPO (Bottom right) on Ant-v2.

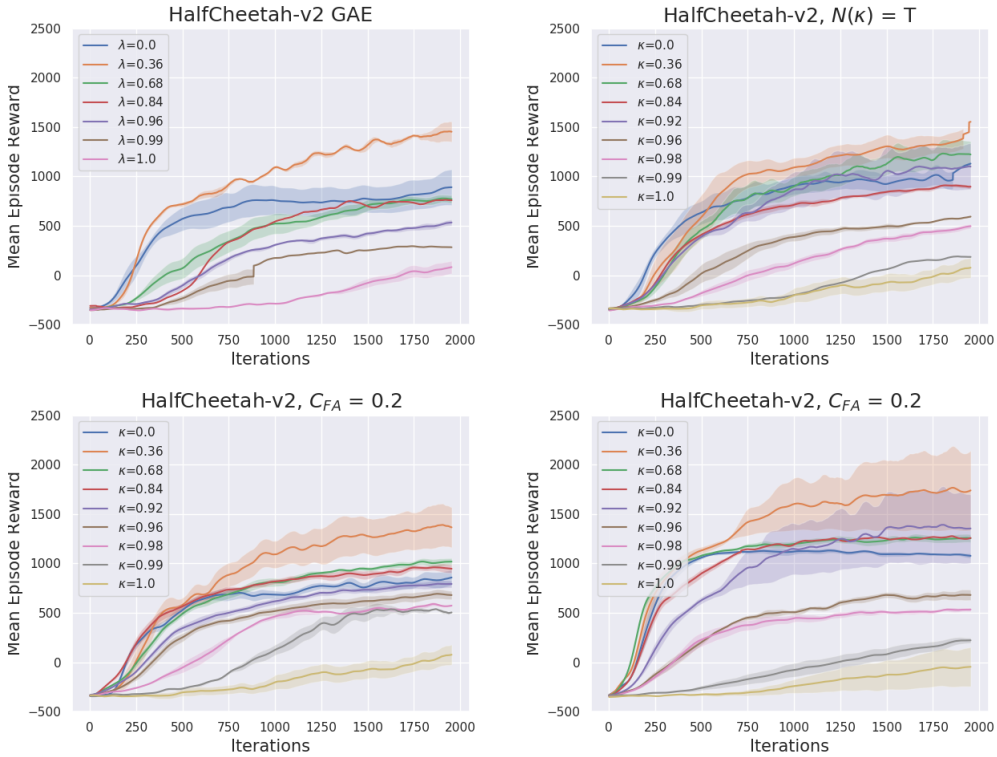

Figure 10: Performance of GAE and 'Naive' baseline (Top row) and κ-PI-TRPO (Bottom left) and κ-VI-TRPO (Bottom right) on HalfCheetah-v2.

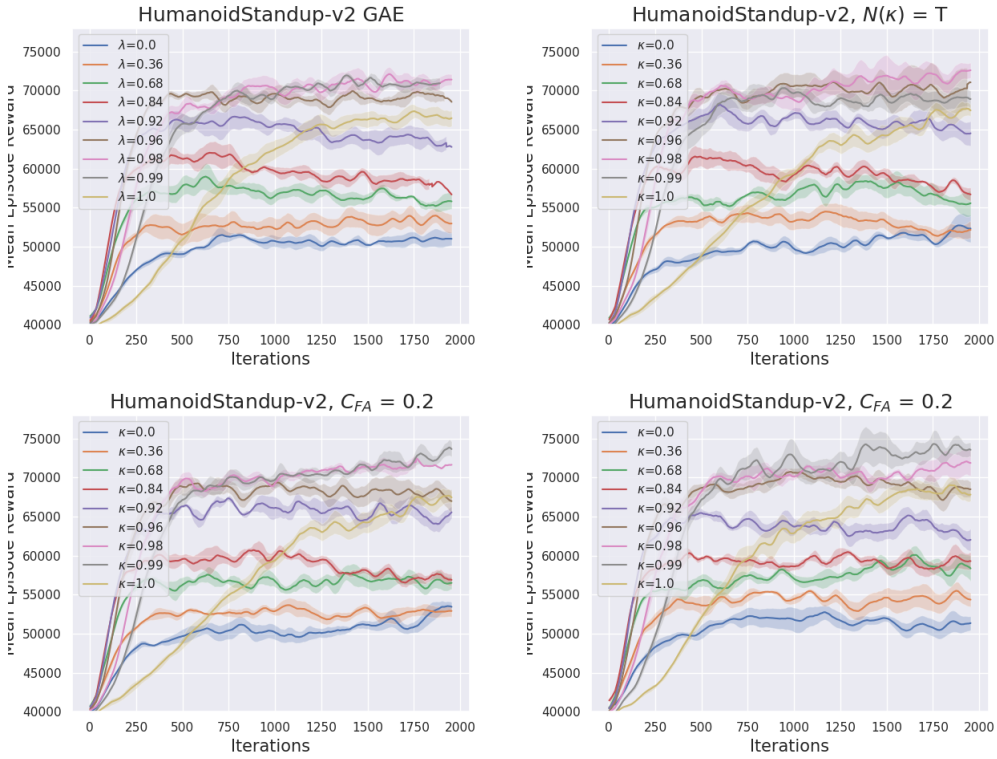

Figure 11: Performance of GAE and 'Naive' baseline (Top row) and κ-PI-TRPO (Bottom left) and κ-VI-TRPO (Bottom right) on HumanoidStandup-v2.

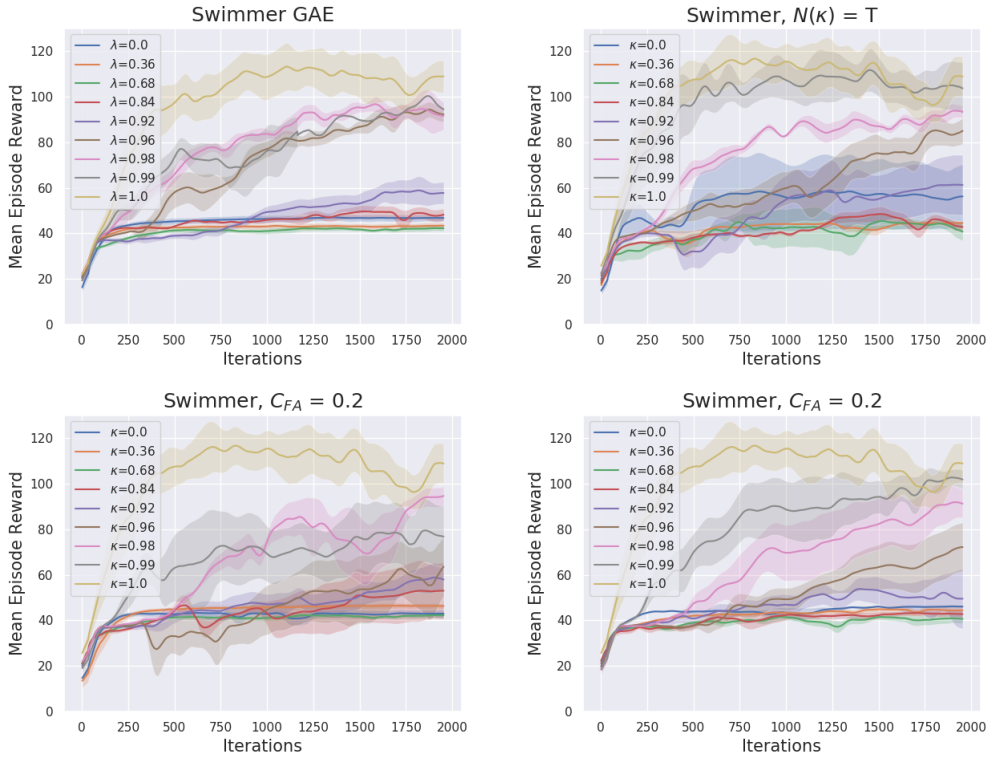

Figure 12: Performance of GAE and 'Naive' baseline (Top row) and κ-PI-TRPO (Bottom left) and κ-VI-TRPO (Bottom right) on Swimmer-v2.