Multi-step Greedy Reinforcement Learning Algorithms

Manan Tomar 1  Yonathan Efroni 2  Mohammad Ghavamzadeh 3

Abstract
Multi-step greedy policies have been extensively used in model-based reinforcement learning (RL), both when a model of the environment is available (e.g., in the game of Go) and when it is learned. In this paper, we explore their benefits in model-free RL, when employed using multi-step dynamic programming algorithms: \(\kappa\)-Policy Iteration (\(\kappa\)-PI) and \(\kappa\)-Value Iteration (\(\kappa\)-VI). These methods iteratively compute the next policy (\(\kappa\)-PI) and value function (\(\kappa\)-VI) by solving a surrogate decision problem with a shaped reward and a smaller discount factor. We derive model-free RL algorithms based on \(\kappa\)-PI and \(\kappa\)-VI in which the surrogate problem can be solved by any discrete or continuous action RL method, such as DQN and TRPO. We identify the importance of a hyper-parameter that controls the extent to which the surrogate problem is solved and suggest a way to set this parameter. When evaluated on a range of Atari and MuJoCo benchmark tasks, our results indicate that for the right range of \(\kappa\), our algorithms outperform DQN and TRPO. This shows that our multi-step greedy algorithms are general enough to be applied over any existing RL algorithm and can significantly improve its performance.

1. Introduction
Reinforcement learning (RL) algorithms solve sequential decision-making problems through repeated interaction with the environment. By incorporating deep neural networks into RL algorithms, the field has recently witnessed remarkable empirical success (e.g., [Mnih et al. 2015, Lillicrap et al. 2015, Silver et al. 2015, 2016, 2017]). Much of this success has been achieved by model-free RL algorithms, such as Q-learning and policy gradients. These algorithms are known to suffer from high variance in their estimations (Greensmith et al. 2004) and to have difficulties in handling function approximation (e.g., [Thrun & Schwartz 1995, Van Hasselt et al. 1995, 2016, 2018]). These issues are intensified in decision problems with long horizon, i.e., when the discount factor, \(\gamma\), is large. Although using smaller values of \(\gamma\) addresses the discount factor-dependent issues and leads to more stable algorithms (Petrov & Scherrer 2009, Jiang et al. 2015), it does not come for free, as the algorithm may return a biased solution, i.e., it may not converge to an optimal (or good) solution for the original decision problem (the one with large value of \(\gamma\)).

Efroni et al. (2018a) recently proposed another approach to mitigate the \(\gamma\)-dependant instabilities in RL in which they study multi-step greedy versions of the well-known Dynamic Programming (DP) algorithms: Policy Iteration (PI) and Value Iteration (VI) (Bertsekas & Tsitsiklis 1996). They also proposed an alternative formulation of the multi-step greedy policy, called \(\kappa\)-greedy policy, and studied the convergence of the resulted PI and VI algorithms: \(\kappa\)-PI and \(\kappa\)-VI. These algorithms iteratively solve a \(\gamma\)-discounted decision problem, 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 \(\gamma\) (discussed above), by iteratively solving decision problems with a shorter \(\gamma\)-horizon, the \(\kappa\)-PI and \(\kappa\)-VI algorithms could converge to an optimal policy of the original decision problem. In this paper, we derive model-free RL algorithms based on the \(\kappa\)-greedy formulation of multi-step greedy policies. As mentioned earlier, the main component of this formulation is (approximately) solving a surrogate decision problem with a shaped reward and a smaller discount factor. Our algorithms build on \(\kappa\)-PI and \(\kappa\)-VI, and solve the surrogate decision problem with the popular deep RL algorithms: Deep Q-Network (DQN) (Mnih et al. 2015) and Trust Region Policy Optimization (TRPO) (Schulman et al. 2015). We call the resulting algorithms \(\kappa\)-PI-DQN, \(\kappa\)-VI-DQN, \(\kappa\)-PI-TRPO, and \(\kappa\)-VI-TRPO, and empirically evaluate and compare them with DQN, TRPO, and Generalized Advantage Estimation (GAE) (Schulman et al. 2016) on Atari (Bellemare et al. 2013) and MuJoCo (Todorov et al. 2012) benchmarks. Our results indicate that for the right range of \(\kappa\), our algorithms outperform DQN and TRPO. This suggests that the performance of these two deep RL algorithms can be improved by applying \(\kappa\)-PI and \(\kappa\)-VI algorithms instead of \(\kappa\)-PI and \(\kappa\)-VI.
Multi-step Greedy Reinforcement Learning Algorithms

improved by using them as a solver within the multi-step greedy PI and VI schemes.

Moreover, our results indicate that the success of our algorithms depends on a number of non-trivial design choices. In particular, we identify the importance of a hyper-parameter that controls the extent to which the surrogate decision problem is solved, and use the theory of multi-step greedy DP to derive a recipe for setting this parameter. We show the advantage of using hard over soft updates, verifying the theory in Efroni et al. (2018b) Thm. 1. By hard and soft update, we refer to fully solving the surrogate MDP in a model-free manner and then evaluating the resulting policy (policy improvement and evaluation steps are separated) v.s. changing the policy at each iteration (policy improvement and evaluation steps are concurrent – each improvement is followed by an evaluation).

We also establish a connection between our multi-step greedy algorithms and GAE. In particular, we show that our κ-PI-TRPO algorithm coincides with GAE and we can obtain GAE by minor modifications to κ-PI-TRPO. Finally, we show the advantage of using our multi-step greedy algorithms over lowering the discount factor in DQN (value-based) and TRPO (policy-based) algorithms. Our results indicate that while lowering the discount factor is detrimental to performance, our multi-step greedy algorithms indeed improve over DQN and TRPO.

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

\begin{equation}
(\mathcal{T}V)(s) = \max_{a \in A} E[r(s_0, a) + \gamma V(s_1) | s_0 = s],
\end{equation}

\begin{equation}
\pi_{\text{1-step}}(s) \in \arg \max_{a \in A} E[r(s_0, a) + \gamma V(s_1) | s_0 = s].
\end{equation}

It is known that $\mathcal{T}$ is a $\gamma$-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 $\pi^*$. 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 $\pi_N$, whose difference with the optimal value is bounded (see e.g., Scherrer (2014) for API):

\begin{equation}
\eta(\pi^*) - \eta(\pi_N) \leq C\delta/(1-\gamma)^2 + \gamma^N V_{\text{max}},
\end{equation}

In Eq. 3 the scalar $\eta(\pi) = E_{s \sim \mu}[V^\pi(s)]$ is the expected value function at the initial state $\mu$ 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 ($\mu$ in Eq. 3), depending 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. $\kappa$-Greedy Policy: $\kappa$-PI and $\kappa$-VI Algorithms

The optimal Bellman operator $\mathcal{T}$ (Eq. 1) and 1-step greedy policy $\pi_{\text{1-step}}$ (Eq. 2) can be generalized to their multi-step versions. The most straightforward form of this generalization is realized by replacing $\mathcal{T}$ and $\pi_{\text{1-step}}$ with $h$-optimal Bellman operator and $h$-step greedy policy (i.e., a lookahead of horizon $h$), respectively. This is done by substituting the 1-step return in Eqs. (1) and (2) with $r(s_0, a) + \gamma^h V(s_h)$, with the $h$-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 for the multi-step optimal Bellman operator and greedy policy, called $\kappa$-optimal Bellman oper-
ator, \( T_\kappa \), and \( \kappa \)-greedy policy, \( \pi_\kappa \), for \( \kappa \in [0, 1] \), i.e.,
\[
(T_\kappa V)(s) = \max_\pi \mathbb{E}[\sum_{t=0}^{\infty} (\gamma \kappa)^t r_t(\kappa, V) \mid s_0 = s, \pi],
\]
\[
\pi_\kappa(s) = \arg \max_\pi \mathbb{E}[\sum_{t=0}^{\infty} (\gamma \kappa)^t r_t(\kappa, V) \mid s_0 = s, \pi],
\]
for all \( s \in \mathcal{S} \). In Eqs. (4) and (5) the shaped reward \( r_t(\kappa, V) \) w.r.t. the value function \( V \) is defined as
\[
r_t(\kappa, V) = r_t + (1 - \kappa) \gamma V(s_{t+1}).
\]
It can be shown that the \( \kappa \)-greedy policy w.r.t. the value function \( V \) is the optimal policy w.r.t. a \( \kappa \)-weighted geometric average of all future \( h \)-step returns (from \( h = 0 \) to \( \infty \)). This can be interpreted as TD(\( \lambda \)) (Sutton & Barto, 2018a) showed that \( \kappa \)-PI and \( \kappa \)-VI converge with a faster exponential rate \( \xi_k = \frac{(1-\kappa)}{1+\kappa \gamma} \leq \gamma \), i.e., \( \|V^* - V^{\pi_k}\|_\infty \leq O(\xi_k N_k) \), with the cost that each iteration of these algorithms is computationally more expensive than that of PI and VI. Finally, we state the following property of \( \kappa \)-PI and \( \kappa \)-VI policies that we use in our \( \kappa \)-PI and \( \kappa \)-VI based RL algorithms described in Section 4.

Asymptotic performance depends on \( \kappa \). Efroni et al. (2018b) Thm. 5) proved the following bound on the performance of \( \kappa \)-PI that is similar to the one in Eq. (3) for API:
\[
\eta(\pi^*) - \eta(\pi_{N_k}) \leq C_k \frac{\delta_k}{(1 - \gamma)^2} + \xi_k N_k V_{\max},
\]
where \( \delta_k \) and \( C_k \) are quantities similar to \( \delta \) and \( C \) in Eq. (3). Note that while the second term on the RHS of Eq. (7) decays with \( N_k \), the first one is independent of \( N_k \).

4. \( \kappa \)-PI and \( \kappa \)-VI based RL Algorithms

As described in Section 3 implementing \( \kappa \)-PI and \( \kappa \)-VI requires iteratively solving a \( \gamma \kappa \)-discounted surrogate MDP with a shaped reward. If a model of the problem is given, the surrogate MDP can be solved using a DP algorithm (see Efroni et al. 2018b Sec. 7). When a model is not available, we should approximately solve the surrogate MDP using a model-free RL algorithm. In this paper, we focus on the latter case and propose RL algorithms inspired by \( \kappa \)-PI and \( \kappa \)-VI. In our algorithms, we use model-free RL algorithms DQN (Mnih et al., 2015) and TRPO (Schulman et al., 2015) (for discrete and continuous action problems, respectively) as subroutines for estimating a \( \kappa \)-greedy policy (Line 4 in Alg. 4 \( \kappa \)-PI, and Line 5 in Alg. 5 \( \kappa \)-VI) and an optimal value of the surrogate MDP (Line 3 in Alg. 4 \( \kappa \)-PI, Line 3 in Alg. 5 \( \kappa \)-VI). We refer to the resulting algorithms as \( \kappa \)-PI-DQN, \( \kappa \)-VI-DQN, \( \kappa \)-PI-TRPO, and \( \kappa \)-VI-TRPO.

In order to have an efficient implementation of our \( \kappa \)-PI and \( \kappa \)-VI based algorithms, the main question to answer is how a fixed number of samples \( T \) should be allocated to different parts of the \( \kappa \)-PI and \( \kappa \)-VI algorithms? More precisely, how shall we set \( N_k \in \mathbb{N} \), i.e., the total number of iterations of our algorithms, and determine the number of samples to solve the surrogate MDP at each iteration? To answer these questions, we devise a heuristic approach based on the theory of \( \kappa \)-PI and \( \kappa \)-VI algorithms, and in particular Eq. (7). Since \( N_k \) only appears explicitly in the second term on the RHS of Eq. (7), an appropriate choice of \( N_k \) is such that \( C_k \delta_k/(1 - \gamma)^2 \approx \xi_k N_k V_{\max} \). Note that setting \( N_k \) to a higher value would not significantly improve the 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:

$$\xi N_\kappa = C_{\text{FA}},$$

(8)

where $C_{\text{FA}}$ is a hyper-parameter that depends on the final-accuracy we aim for. For example, if our goal is to obtain 90% accuracy, we would set $C_{\text{FA}} = 0.1$, which results in $N_\kappa = 0.99 \approx 4$ and $N_\kappa = 0.5 \approx 115$, for $\gamma = 0.99$. Our experimental results in Section 5 suggest that this approach leads to a reasonable choice for the total number of iterations $N_\kappa$. It is important to note the following facts: 1) as we increase $\kappa$, we expect less iterations are needed for $\kappa$-PI and $\kappa$-VI to converge to a good policy, and 2) the effective horizon[†] of the surrogate MDP that $\kappa$-PI and $\kappa$-VI solve at each iteration increases with $\kappa$.

Lastly, we need to determine the number of samples for each iteration of our $\kappa$-PI and $\kappa$-VI based algorithms. 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

$$T_\kappa = T/N_\kappa.$$

In the rest of the paper, we first derive our DQN-based and TRPO-based algorithms in Sections 4.1 and 4.2. It is important to note that for $\kappa = 1$, our algorithms are reduced to DQN and TRPO. We then conduct a set of experiments with our algorithms in Sections 5.1 and 5.2 in which we carefully study the effect of $\kappa$ and $N_\kappa$ (or equivalently the hyper-parameter $C_{\text{FA}}$, defined by Eq. 8) on their performance.

4.1. $\kappa$-PI-DQN and $\kappa$-VI-DQN Algorithms

Algorithm[3] presents the pseudo-code of $\kappa$-PI-DQN. Due to space constraints, we report the detailed pseudo-code in Appendix A.1 (Alg. 5). We use four neural networks in this algorithm, two to represent the $Q$-function of the original MDP (with discount factor $\gamma$ and reward $r$), $Q_\phi$ ($Q$-network) and $Q'_\phi$ (target network), and two for the $Q$-function of the surrogate MDP, $Q_\theta$ ($Q$-network) and $Q'_\theta$ (target network). In the policy improvement step, we use DQN to solve the $\gamma_\kappa$-discounted surrogate MDP with the shaped reward $r_j(\kappa, V_\phi) = r_j + \gamma(1-\kappa)V_\phi(s_{j+1})$, i.e., $M_{\gamma_\kappa}(V_\phi)$, where $V_\phi \simeq V^{\pi_{\kappa-1}}$ and is computed as $V_\phi(s) = Q_\phi(s, \arg \max_a Q'_\phi(s, a))$. The output of DQN is (approximately) the optimal $Q$-function of $M_{\gamma_\kappa}(V_\phi)$, and thus, the new policy $\pi_s$, which is (the approximate) $\kappa$-greedy policy w.r.t. $V_\phi$ is equal to $\arg \max_a Q'_\phi(s, a)$. In the policy evaluation step, we use off-policy TD(0) to evaluate the $Q$-function of the current policy $\pi_s$, i.e., $Q_\phi \simeq Q^\pi$. Although what is needed is an estimate of the value function

\[ \frac{1}{(1-\gamma)} \]

of the current policy, $V_\phi \simeq V^{\pi_s}$, we chose to evaluate its $Q$-function, because the available data (the transitions stored in the replay buffer) is off-policy, and unlike the value function, the $Q$-function of a fixed policy can be easily evaluated with this type of data using off-policy TD(0).

**Remark 1.** In the policy improvement phase, $\pi_{\kappa-1}$ is computed as the $\arg \max$ of $Q'_\phi(s, a)$ (Line 10), a quantity that is not constant and is (slowly) changing during this phase. Addressing this issue requires using an additional target network that is set to $Q_\phi$ only at the end of each improvement step and its $\arg \max$ is used to compute $\pi_{\kappa-1}$ throughout the improvement step of the next iteration. We tested using this additional network in our experiments, but it did not improve the performance, and thus, we decided to report the algorithm without it.

We report the pseudo-code of $\kappa$-VI-DQN in Appendix A.1 (Alg. 6). Note that $\kappa$-VI simply repeats $V \leftarrow T_\kappa V$ and computes $T_\kappa V$, which is the optimal value of the surrogate MDP $M_{\gamma_\kappa}(V)$. In $\kappa$-VI-DQN, we repeatedly solve $M_{\gamma_\kappa}(V)$ by DQN and use its (approximately) optimal $Q$-function to shape the reward of the next iteration. The algorithm uses three neural networks, two to solve the surrogate MDP by DQN, $Q_\phi$ (network) and $Q'_\phi$ (target network), and one to store its optimal $Q$-function to use it for the shaped reward in the next iteration, $Q_\phi$. Let $Q^*_{\gamma_\kappa,V}$ and $V^*_{\gamma_\kappa,V}$ be the optimal $Q$ and $V$ functions of $M_{\gamma_\kappa}(V)$. 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 $M_{\gamma_\kappa}(V)$ (Sec. 3).

Therefore, in $\kappa$-VI-DQN, we shape the reward at each iteration by $\max_a Q'_\phi(s, a)$, where $Q_\phi$ is the output of the DQN

\[ \frac{1}{(1-\gamma)} \]
Algorithm 4 \(\kappa\)-PI-TRPO

1. Initialize V-networks \(\theta\) and \(\phi\); policy network \(\pi\);
2. for \(i = 0, \ldots, N_\kappa - 1\) do
3. for \(t = 1, \ldots, T\) do
4. Simulate the current policy \(\pi_i\) for \(M\) steps and calculate the following return for all steps \(j\):
   \(R_j(\kappa, \theta_i) = \sum_{\tau=0}^{M} \gamma^{\tau-j} r_j(\kappa, \theta_i)\) and \(\rho_j = \sum_{\tau=1}^{M} \gamma^{\tau-j} r_i(\kappa, \theta_i)\);
5. Update \(\theta\) by minimizing the batch loss function:
   \(\mathcal{L}_{\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, \theta_i))^2\);
6. # Policy Improvement
7. Update \(\psi\) using TRPO and the batch \(\{(R_j(\kappa, \theta_i), V_\theta(s_j))\}_{j=1}^{N}\);
8. end for
9. # Policy Evaluation
10. Update \(\phi\) by minimizing the batch loss function:
    \(\mathcal{L}_{\phi} = \frac{1}{\gamma} \sum_{j=1}^{\gamma} (\phi_j(s_j) - \rho_j)^2\);
11. end for

from the previous iteration, i.e., \(\max_a Q_\theta(s, a) \simeq T_\kappa V_{i-1}\).

4.2. \(\kappa\)-PI-TRPO and \(\kappa\)-VI-TRPO Algorithms

Algorithm 4 presents the pseudo-code of \(\kappa\)-PI-TRPO (a detailed pseudo-code is reported in Appendix B.1, Alg. 7). Recall that TRPO iteratively updates the current policy using its return and an estimate of its value function. At each iteration \(i\) of \(\kappa\)-PI-TRPO: 1) we use the estimate of the value of the current policy \(\hat{V}_\theta \simeq V^{\pi_{i-1}}\) to calculate the return \(R(\kappa, \theta)\) and the estimate of the value function \(\hat{V}_\theta\) of the surrogate MDP \(M_{\gamma, V}(\theta)\), 2) we use \(R(\kappa, \theta)\) and \(\hat{V}_\theta\) to compute the new policy \(\pi_i\) (TRPO style), and 3) we estimate the value of the new policy \(\hat{V}_\theta \simeq V^{\pi_i}\) in the original MDP (with discount factor \(\gamma\) and reward \(r\)). The algorithm uses three neural networks, one for the value function of the original MDP, \(\hat{V}(s)\) one for the value function of the surrogate MDP, \(\hat{V}(s)\), and one for the policy, \(\pi\).

We report the pseudocode of \(\kappa\)-VI-TRPO in Appendix B.1, Alg. 8. As previously noted, \(\kappa\)-VI iteratively solves the surrogate MDP and uses its optimal value \(T_\kappa V_{i-1}\) to shape the reward of the surrogate MDP in the next iteration. In \(\kappa\)-VI-TRPO, we solve the surrogate MDP \(M_{\gamma, V}(V_{i-1} = \theta)\) with TRPO until its policy \(\pi_{i}\) converges to the optimal policy of \(M_{\gamma, V}(V_{i-1} = \theta)\) and its value \(\hat{V}_\theta\) converges to \(T_\kappa V_{i-1} = T_\kappa \hat{V}_\theta\). We then replace \(V_i\) with \(V_\theta(V_i = \theta, \theta)\) and repeat this process.

5. Experimental Results

In our experiments, we specifically focus on answering the following questions:

1. Does the performance of DQN and TRPO improve when using them as \(\kappa\)-greedy solvers in \(\kappa\)-PI and \(\kappa\)-VI? Is there a performance tradeoff w.r.t. to \(\kappa\)?
Multi-step Greedy Reinforcement Learning Algorithms

2. Following $\kappa$-PI and $\kappa$-VI, our DQN and TRPO implementations of these algorithms devote a significant $T_\kappa$ number of samples 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 $\kappa$?

5.1. $\kappa$-PI-DQN and $\kappa$-VI-DQN Experiments

In this section, we empirically analyze the performance of the $\kappa$-PI-DQN and $\kappa$-VI-DQN algorithms on the Atari domains: Breakout, SpaceInvaders, Seaquest, Enduro, BeamRider and Qbert (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). The total number of samples is set 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 $\kappa$-PI-DQN and $\kappa$-VI-DQN for several $\kappa$ values. In both algorithms, the best performance was obtained with $C_{FA} = 0.05$. Therefore, $C_{FA}$ is set to 0.05 for all our experiments with other Atari domains.

Figure 3 and 4 show the training performance of $\kappa$-PI-DQN and $\kappa$-VI-DQN on Breakout and SpaceInvaders for the best value of $C_{FA} = 0.05$, as well as for the ‘naive’ baseline $N_\kappa = T$ and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO for $C_{FA} = 0.2$ on Walker. See Appendix B.2 for performance w.r.t. different $C_{FA}$ values. Each iteration corresponds to roughly 1000 environment samples, and thus the total training samples is to 2 million.

2. Following $\kappa$-PI and $\kappa$-VI, our DQN and TRPO implementations of these algorithms devote a significant $T_\kappa$ number of samples 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 $\kappa$?

5.2. $\kappa$-PI-TRPO and $\kappa$-VI-TRPO Experiments

In this section, we empirically analyze the performance of the $\kappa$-PI-TRPO and $\kappa$-VI-TRPO algorithms on the MuJoCo (Todorov et al., 2012) based OpenAI Gym domains:
Multi-step Greedy Reinforcement Learning Algorithms

<table>
<thead>
<tr>
<th>Domain</th>
<th>Alg.</th>
<th>$\kappa$-beat</th>
<th>$\kappa = 0$</th>
<th>$\kappa = 1$</th>
<th>$N_\kappa = T, \kappa$-beat ($\kappa$-PI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>$\kappa$-PI</td>
<td>222±77, $\kappa = 0.84$</td>
<td>154±3</td>
<td>134±4</td>
<td>170±2, $\kappa = 0.68$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>181±77, $\kappa = 0.68$</td>
<td>174±5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpaceInv.</td>
<td>$\kappa$-PI</td>
<td>755±231, $\kappa = 0.84$</td>
<td>613±20</td>
<td>656±17</td>
<td>700±21, $\kappa = 0.98$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>712±25, $\kappa = 0.92$</td>
<td>687±32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seaquest</td>
<td>$\kappa$-PI</td>
<td>5159±2922, $\kappa = 0.92$</td>
<td>2612±238</td>
<td>3099±191</td>
<td>3897±218, $\kappa = 0.68$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>3253±402, $\kappa = 0.84$</td>
<td>2680±382</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enduro</td>
<td>$\kappa$-PI</td>
<td>533±121, $\kappa = 0.84$</td>
<td>478±10</td>
<td>224±110</td>
<td>535±131, $\kappa = 0.68$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>486±23, $\kappa = 0.84$</td>
<td>443±90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BeamRider</td>
<td>$\kappa$-PI</td>
<td>3849±1100, $\kappa = 1.0$</td>
<td>3103±279</td>
<td>3849±1100</td>
<td>3849±1100, $\kappa = 1.0$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>4277±1209, $\kappa = 0.84$</td>
<td>3714±433</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qbert</td>
<td>$\kappa$-PI</td>
<td>8157±265, $\kappa = 0.84$</td>
<td>6719±520</td>
<td>7563±598</td>
<td>7968±218, $\kappa = 0.98$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>8060±1350, $\kappa = 0.84$</td>
<td>7258±385</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The final training performance of $\kappa$-PI-DQN and $\kappa$-VI-DQN on the Atari domains, for the hyper-parameter $C_{FA} = 0.05$. The values are reported for a 95% confidence interval across 10 random runs (empirical mean ± 1.96 × empirical standard deviation / √n = 10). The best scores are in bold and multiple bold values for a domain denote an insignificant statistical difference between them.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Alg.</th>
<th>$\kappa$-beat</th>
<th>$\kappa = 0$</th>
<th>$\kappa = 1$</th>
<th>$N_\kappa = T, \kappa$-beat ($\kappa$-PI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walker</td>
<td>$\kappa$-PI</td>
<td>1438±188, $\kappa = 0.68$</td>
<td>1371±192</td>
<td>594±99</td>
<td>1082±1100, $\kappa = 0.0$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>954±885, $\kappa = 0.68$</td>
<td>830±262</td>
<td>1090±99</td>
<td>1094±139, $\lambda = 0.0$</td>
</tr>
<tr>
<td>Ant</td>
<td>$\kappa$-PI</td>
<td>1377±1833, $\kappa = 0.68$</td>
<td>1006±106</td>
<td>-19±3</td>
<td>1195±213, $\kappa = 0.36$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>2998±2940, $\kappa = 0.68$</td>
<td>1879±128</td>
<td></td>
<td>1322±213, $\lambda = 0.36$</td>
</tr>
<tr>
<td>HalfCheetah</td>
<td>$\kappa$-PI</td>
<td>1334±1311, $\kappa = 0.36$</td>
<td>907±176</td>
<td>-18±37</td>
<td>6814±10, $\kappa = 0.98$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>1457±1461, $\kappa = 0.36$</td>
<td>1072±130</td>
<td></td>
<td>71331±1488, $\kappa = 1.0$</td>
</tr>
<tr>
<td>HumanoidStand</td>
<td>$\kappa$-PI</td>
<td>72604±12199, $\kappa = 0.99$</td>
<td>52936±1529</td>
<td>71932±21228, $\lambda = 0.98$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>72821±9088, $\kappa = 0.99$</td>
<td>51148±1377</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swimmer</td>
<td>$\kappa$-PI</td>
<td>107±126, $\kappa = 1.0$</td>
<td>43±3</td>
<td>107±126</td>
<td>103±138, $\lambda = 1.0$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>114±183, $\kappa = 1.0$</td>
<td>46±1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hopper</td>
<td>$\kappa$-PI</td>
<td>1486±324, $\kappa = 0.68$</td>
<td>1012±263</td>
<td>1142±1411</td>
<td>1434±1298, $\kappa = 0.98$</td>
</tr>
<tr>
<td></td>
<td>$\kappa$-VI</td>
<td>1069±786, $\kappa = 0.92$</td>
<td>531±125</td>
<td></td>
<td>1600±1384, $\lambda = 0.84$</td>
</tr>
</tbody>
</table>

Table 2: The final training performance of $\kappa$-PI-TRPO and $\kappa$-VI-TRPO on the MuJoCo domains, for the hyper-parameter $C_{FA} = 0.2$. The values are reported for a 95% confidence interval across 10 random runs (empirical mean ± 1.96 × empirical standard deviation / √n = 10). The best scores are in bold and multiple bold values for a domain denote an insignificant statistical difference between them.

Walker2d-v2, Ant-v2, HalfCheetah-v2, HumanoidStandup-v2, Swimmer-v2, and Hopper-v2 (Brockman et al. 2016). As in Section 5.1, 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 $T \approx 2 \times 10^6$. This is the number of samples after which our TRPO-based algorithms approximately converge. For each value of $C_{FA}$, we test $\kappa$-PI-TRPO and $\kappa$-VI-TRPO for several $\kappa$ values. In both algorithms, the best performance was obtained with $C_{FA} = 0.2$, and thus, we set $C_{FA} = 0.2$ in our experiments with other MuJoCo domains.

Figure 2 and 3 show the training performance of $\kappa$-PI-TRPO and $\kappa$-VI-TRPO on Walker and Ant domains for the best value of $C_{FA} = 0.2$, as well as for the ‘naive’ baseline $T_\kappa = 1$, or equivalently $N_\kappa = T$. The results on Walker for the other $C_{FA}$ values and the other MuJoCo domains (with $C_{FA} = 0.2$) are reported in Appendix B.2 and B.3.

Table 2 shows the final training performance of $\kappa$-PI-TRPO and $\kappa$-VI-TRPO on the MuJoCo domains with $C_{FA} = 0.2$. The results exhibit both $\kappa$-PI-TRPO and $\kappa$-VI-TRPO yield better performance than TRPO ($\kappa = 1$). Furthermore, they show that the algorithms with $C_{FA} = 0.2$ perform better than with $N_\kappa = T$ for three out of six domains (Walker, Ant and HalfCheetah) and equally well for the remaining three (HumanoidStandup, Swimmer and Hopper).

6. Discussion and Related Work

Comparison with GAE: There is a close connection between the Generalized Advantage Estimation (GAE) algorithm (Schulman et al. 2015) and $\kappa$-PI. In GAE, the policy is updated by the following gradient:

$$\nabla_\theta \mathbb{E}_{s \sim d_{\pi}} [V_{\pi}(s)] = \mathbb{E}_{s \sim d_{\pi}} [\nabla_\theta \log \pi_{\theta}(s) \sum_t (\gamma \lambda)^t \delta_t],$$

$$\delta_t = r_t + \gamma V_{t+1} - V_t,$$

(10)
where \( d_\pi^\mu \) is the occupancy measure of policy \( \pi \). Eq. [10] can be interpreted as a gradient in a \( \gamma \lambda \)-discounted MDP with shaped rewards \( \delta_V \), which we refer to as \( \mathcal{M}_{\gamma \lambda}(\delta_V) \). As noted in Efroni et al. [2018a, Sec. 6], an optimal policy \( \pi^*_{\gamma \lambda} \) of the MDP \( \mathcal{M}_{\gamma \lambda}(\delta_V) \) is also optimal in \( \mathcal{M}_{\gamma \kappa}(V) \) with \( \kappa = \lambda \). This means that \( \pi^*_{\gamma \lambda} \) is the \( \kappa \)-greedy policy w.r.t. \( V \).

Comparing the two algorithms, we notice that GAE is conceptually similar to \( \kappa \)-PI-TRPO with \( T_\kappa = 1 \), or equivalently \( N_\kappa = T \). In fact, in the case of \( T_\kappa = 1 \), we can obtain a pseudocode of GAE by removing Line 5 (no need to estimate the value of the surrogate MDP) and replacing \( r_t(\kappa, V_\phi) \) with the TD-error of the original MDP on Line 4 in Algorithm [4].

In Section 5.2, we compared the empirical performance of GAE with that of \( \kappa \)-PI-TRPO and \( \kappa \)-VI-TRPO (see Figure 5 and Table 2). The results show that \( \kappa \)-PI-TRPO and \( \kappa \)-VI-TRPO perform better than or on par with GAE. Moreover, we observe that in most domains, the performance of GAE is equivalent to that of \( \kappa \)-PI-TRPO with \( N_\kappa = T \). This is in accordance with the description above connecting GAE to our naive baseline implementation. We report all GAE results in Appendix B.3.

Remark 2 (GAE Implementation). In the OpenAI implementation of GAE, the value network is updated w.r.t. to the target \( \sum_t (\gamma \lambda)^t r_t \), whereas in the GAE paper (Schulman et al., 2016), \( \sum_t \gamma^t r_t \) is used as the target. We chose the latter form in our implementation of GAE to be in accord with the paper.

Lowering Discount Factor in DQN and TRPO: To show the advantage of \( \kappa \)-PI and \( \kappa \)-VI based algorithms over simply lowering the discount factor \( \gamma \), we test the performance of the “vanilla” DQN and TRPO algorithms with values of \( \gamma \) lower than the one previously used (i.e., \( \gamma = 0.99 \)). As evident from Figure 5 only in the Ant domain, this approach results in an improved performance (for \( \gamma = 0.68 \)). On the other hand, in the Ant domain, the performance of \( \kappa \)-PI-TRPO, and especially \( \kappa \)-VI-TRPO, surpasses that of TRPO with the lower value of \( \gamma = 0.68 \). In Breakout, SpaceInvaders, and Walker, the performance of DQN and TRPO worsens or remains unchanged when we lower the discount factor (DQN and TRPO do not benefit from lowering \( \gamma \)), while our \( \kappa \)-PI and \( \kappa \)-VI based algorithms perform better with lowering the value of \( \kappa \) (note that our algorithms are reduced to DQN and TRPO for \( \kappa = 1 \)).

Remark 3. While we observe better performance for smaller \( \gamma \) values in some MuJoCo domains, e.g., \( \gamma = 0.68 \) in Ant, lowering \( \gamma \) always results in inferior performance in the Atari domains. This is due to the fact that a number of MuJoCo domains, such as Ant, are inherently short-horizon decision problems, and thus, their performance does not degrade (even sometimes improves) with lowering the discount factor. On the other hand, the Atari problems are generally not short-horizon, and thus, their performance degrades with lowering \( \gamma \).
7. Conclusion and Future Work

In this paper, we studied the use of multi-step greedy policies in model-free RL and showed that in most problems, the algorithms derived from this formulation achieve a better performance than their single-step counterparts. We adopted the $\kappa$-greedy formulation of multi-step greedy policies (Efroni et al., 2018a) and derived four model-free RL algorithms. The main component of the policy and value iteration algorithms derived from this formulation, $\kappa$-PI and $\kappa$-VI, is solving a surrogate decision problem with a shaped reward and a smaller discount factor. Our algorithms use popular deep RL algorithms, DQN and TRPO, to solve the surrogate decision problem, and thus, we refer to them as $\kappa$-PI-DQN, $\kappa$-VI-DQN, $\kappa$-PI-TRPO, and $\kappa$-VI-TRPO. We empirically evaluated our proposed algorithms and compared them with DQN, TRPO, and GAE on Atari and MuJoCo benchmarks. Our experiments show that for a large range of $\kappa$, our algorithms perform better than DQN and TRPO. Furthermore, we proposed a recipe to allocate the total sample budget to the evaluation and improvement phases of our algorithms, and empirically demonstrated the importance of this allocation. We also showed how GAE can be derived by minor modifications to $\kappa$-PI-TRPO, and thus, is a $\kappa$-greedy RL algorithm. Finally, we showed the advantage of multi-step greedy formulation over lowering the discount factor in DQN and TRPO. Our results indicate that while the performance of DQN and TRPO degrades with lowering the discount factor, our multi-step greedy algorithms improve over DQN and TRPO.

An interesting future direction would be to use other multi-step greedy formulations (Bertsekas & Tsitsiklis, 1996; Bertsekas, 2018; Efroni et al., 2018a; Sun et al., 2018; Shan et al., 2019) to derive model-free RL algorithms. Another direction is to use multi-step greedy in model-based RL (e.g., Kumar et al., 2016; Talvitie, 2017; Luo et al., 2018; Janner et al., 2019) and solve the surrogate decision problem with an approximate model. We conjecture that in this case one may set $\kappa$ – or more generally, the planning horizon – as a function of the quality of the approximate model: gradually increasing $\kappa$ as the approximate model gets closer to the real one. We leave theoretical and empirical study of this problem for future work. Finally, we believe using adaptive $\kappa$ would greatly improve the performance of our proposed algorithms. We leave verifying this and how $\kappa$ should change as a function of errors in gradient and value estimation for future work.

References


