Appendix

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

A.1. Detailed Pseudo-codes

In this section, we report the detailed pseudo-codes of $\kappa$-PI-DQN and $\kappa$-VI-DQN algorithms, described in Section 4.3, side-by-side.

Algorithm 5 $\kappa$-PI-DQN

1: Initialize replay buffer $D$: Q-networks $Q_\theta$ and $Q_\phi$ with random weights $\theta$ and $\phi$;
2: Initialize target networks $Q'_\theta$ and $Q'_\phi$ with weights $\theta' \leftarrow \theta$ and $\phi' \leftarrow \phi$;
3: for $i = 0, \ldots, N_\kappa - 1$ do
4:   # Policy Improvement
5:   for $t = 1, \ldots, T_\kappa$ do
6:     Select $a_t$ as an $\epsilon$-greedy action w.r.t. $Q_\theta(s_t, a)$;
7:     Execute $a_t$, observe $r_t$ and $s_{t+1}$, and store the tuple $(s_t, a_t, r_t, s_{t+1})$ in $D$;
8:     Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $D$;
9:     Update $\theta$ by minimizing the following loss function:
10:    $L_{Q_\theta} = \frac{1}{N} \sum_{j=1}^N \left[ Q_\theta(s_j, a_j) - (r_j + \gamma \max_a Q'_\theta(s_{j+1}, a)) \right]^2$, where
11:    $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi_{t-1}(s_{j+1}))$ and $\pi_{t-1}(s_{j+1}) \in \arg \max_a Q'_\phi(s_{j+1}, a)$;
12:    Copy $\theta$ to $\theta'$ occasionally $(\theta' \leftarrow \theta)$;
13:   end for
14:   # Policy Evaluation
15:   for $t' = 1, \ldots, T(\kappa)$ do
16:     Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $D$;
17:     Update $\phi$ by minimizing the following loss function:
18:    $L_{Q_\phi} = \frac{1}{N} \sum_{j=1}^N \left[ Q_\phi(s_j, a_j) - (r_j + \gamma Q'_\phi(s_{j+1}, \pi_{t}(s_{j+1}))) \right]^2$;
19:    Copy $\phi$ to $\phi'$ occasionally $(\phi' \leftarrow \phi)$;
20:   end for
21: end for

Algorithm 6 $\kappa$-VI-DQN

1: Initialize replay buffer $D$: Q-networks $Q_\theta$ and $Q_\phi$ with random weights $\theta$ and $\phi$;
2: Initialize target network $Q'_\theta$ with weights $\theta' \leftarrow \theta$;
3: for $i = 0, \ldots, N_\kappa - 1$ do
4:   # Evaluate $T_\kappa V_\phi$ and the $\kappa$-greedy policy w.r.t. $V_\phi$
5:   for $t = 1, \ldots, T_\kappa$ do
6:     Select $a_t$ as an $\epsilon$-greedy action w.r.t. $Q_\theta(s_t, a)$;
7:     Execute $a_t$, observe $r_t$ and $s_{t+1}$, and store the tuple $(s_t, a_t, r_t, s_{t+1})$ in $D$;
8:     Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $D$;
9:     Update $\theta$ by minimizing the following loss function:
10:    $L_{Q_\theta} = \frac{1}{N} \sum_{j=1}^N \left[ Q_\theta(s_j, a_j) - (r_j + \kappa \max_a Q'_\theta(s_{j+1}, a)) \right]^2$, where
11:    $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi(s_{j+1}))$ and $\pi(s_{j+1}) \in \arg \max_a Q_\phi(s_{j+1}, a)$;
12:    Copy $\theta$ to $\theta'$ occasionally $(\theta' \leftarrow \theta)$;
13: end for
14: Copy $\theta$ to $\phi$ $(\phi \leftarrow \theta)$
15: end for
Multi-step Greedy Reinforcement Learning Algorithms

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizon (T)</td>
<td>1000</td>
</tr>
<tr>
<td>Adam stepsize</td>
<td>$1 \times 10^{-4}$</td>
</tr>
<tr>
<td>Target network update frequency</td>
<td>1000</td>
</tr>
<tr>
<td>Replay memory size</td>
<td>100000</td>
</tr>
<tr>
<td>Discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>Total training time steps</td>
<td>10000000</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>32</td>
</tr>
<tr>
<td>Initial exploration</td>
<td>1</td>
</tr>
<tr>
<td>Final exploration</td>
<td>0.1</td>
</tr>
<tr>
<td>Final exploration frame</td>
<td>1000000</td>
</tr>
<tr>
<td>#Runs used for plot averages</td>
<td>10</td>
</tr>
<tr>
<td>Confidence interval for plot runs</td>
<td>( \sim 95% )</td>
</tr>
</tbody>
</table>

Table 3: Hyperparameters for \( \kappa \)-PI-DQN and \( \kappa \)-VI-DQN.

A.2. Ablation Test for \( C_{FA} \)

![Figure 4: Performance of \( \kappa \)-PI-DQN and \( \kappa \)-VI-DQN on Breakout for different values of \( C_{FA} \).](image)

A.3. \( \kappa \)-PI-DQN and \( \kappa \)-VI-DQN Plots

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

![Figure 5: Training performance of the ‘naive’ baseline \( N_{\kappa} = T \) and \( \kappa \)-PI-DQN, \( \kappa \)-VI-DQN for \( C_{FA} = 0.05 \) on SpaceInvaders](image)
Figure 6: Training performance of the ‘naive’ baseline $N_r = T$ and $\kappa$-PI-DQN, $\kappa$-VI-DQN for $C_{FA} = 0.05$ on Seaquest

Figure 7: Training performance of the ‘naive’ baseline $N_r = T$ and $\kappa$-PI-DQN, $\kappa$-VI-DQN for $C_{FA} = 0.05$ on Enduro

Figure 8: Training performance of the ‘naive’ baseline $N_r = T$ and $\kappa$-PI-DQN, $\kappa$-VI-DQN for $C_{FA} = 0.05$ on BeamRider

Figure 9: Training performance of the ‘naive’ baseline $N_r = T$ and $\kappa$-PI-DQN, $\kappa$-VI-DQN for $C_{FA} = 0.05$ on Qbert
B. $\kappa$-PI-TRPO and $\kappa$-VI-TRPO Algorithms

B.1. Detailed Pseudo-codes

In this section, we report the detailed pseudo-codes of the $\kappa$-PI-TRPO and $\kappa$-VI-TRPO algorithms, described in Section 4.4, side-by-side.

Algorithm 7 $\kappa$-PI-TRPO

1: Initialize $V$-networks $V_\theta$ and $V_\varphi$ with random weights $\theta$ and $\varphi$; policy network $\pi_\psi$ with random weights $\psi$;
2: for $i = 0, \ldots, N_\kappa - 1$ do
3:   for $t = 1, \ldots, T_\kappa$ do
4:     Simulate the current policy $\pi_\psi$ for $M$ time-steps;
5:     for $j = 1, \ldots, M$ do
6:       Calculate $R_j(\kappa, V_\varphi) = \sum_{t=j}^{M}(\gamma \kappa)^{t-j}r_t(\kappa, V_\varphi)$ and $\rho_j = \sum_{t=j}^{M} \gamma^{t-j}r_t$;
7:   end for
8:   Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated $M$ time-steps;
9:   Update $\theta$ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V_\varphi))^2$;
10: end for
11: Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated $M$ time-steps;
12: Update $\psi$ using TRPO with advantage function computed by $\{(R_j(\kappa, V_\varphi), V_\theta(s_j))\}_{j=1}^{N}$;
13: end for
14: # Policy Evaluation
15: Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated $M$ time-steps;
16: Update $\phi$ by minimizing the loss function: $\mathcal{L}_{V_\phi} = \frac{1}{N} \sum_{j=1}^{N} (V_\phi(s_j) - \rho_j)^2$;
17: end for

Algorithm 8 $\kappa$-VI-TRPO

1: Initialize $V$-networks $V_\theta$ and $V_\varphi$ with random weights $\theta$ and $\varphi$; policy network $\pi_\psi$ with random weights $\psi$;
2: for $i = 0, \ldots, N_\kappa - 1$ do
3:   for $t = 1, \ldots, T_\kappa$ do
4:     Simulate the current policy $\pi_\psi$ for $M$ time-steps;
5:     for $j = 1, \ldots, M$ do
6:       Calculate $R_j(\kappa, V_\varphi) = \sum_{t=j}^{M}(\gamma \kappa)^{t-j}r_t(\kappa, V_\varphi)$;
7:   end for
8:   Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated $M$ time-steps;
9:   Update $\theta$ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V_\varphi))^2$;
10: Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated $M$ time-steps;
11: Update $\psi$ using TRPO with advantage function computed by $\{(R_j(\kappa, V_\varphi), V_\theta(s_j))\}_{j=1}^{N}$;
12: end for
13: Copy $\theta$ to $\phi$ ($\phi \leftarrow \theta$);
14: end for
Multi-step Greedy Reinforcement Learning Algorithms

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizon (T)</td>
<td>1000</td>
</tr>
<tr>
<td>Adam stepsize</td>
<td>$1 \times 10^{-3}$</td>
</tr>
<tr>
<td>Number of samples per Iteration</td>
<td>1024</td>
</tr>
<tr>
<td>Entropy coefficient</td>
<td>0.01</td>
</tr>
<tr>
<td>Discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>Number of Iterations</td>
<td>2000</td>
</tr>
<tr>
<td>Minibatch size</td>
<td>128</td>
</tr>
<tr>
<td>#Runs used for plot averages</td>
<td>10</td>
</tr>
<tr>
<td>Confidence interval for plot runs</td>
<td>$\sim 95%$</td>
</tr>
</tbody>
</table>

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

B.2. Ablation Test for $C_{FA}$

Figure 10: Performance of $\kappa$-PI-TRPO and $\kappa$-VI-TRPO on Walker2d-v2 for different values of $C_{FA}$.

B.3. $\kappa$-PI-TRPO and $\kappa$-VI-TRPO Plots

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

Figure 11: Performance of GAE, ‘Naive’ baseline and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO on Ant-v2.
Figure 12: Performance of GAE, ‘Naive’ baseline and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO on HalfCheetah-v2.

Figure 13: Performance of GAE, ‘Naive’ baseline and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO on HumanoidStandup-v2.

Figure 14: Performance of GAE, ‘Naive’ baseline and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO on Swimmer-v2.

Figure 15: Performance of GAE, ‘Naive’ baseline and $\kappa$-PI-TRPO, $\kappa$-VI-TRPO on Hopper-v2.