Sample-Efficient Model-Free Reinforcement Learning with Off-Policy Critics

Denis Steckelmacher[1] ✉, Hélène Plisnier[1], Diederik M. Roijers[2], and Ann Nowé[1]

[1] Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
[2] VU Amsterdam, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
dsteckel@ai.vub.ac.be

Abstract. Value-based reinforcement-learning algorithms provide state-of-the-art results in model-free discrete-action settings, and tend to outperform actor-critic algorithms. We argue that actor-critic algorithms are limited by their need for an *on-policy* critic. We propose Bootstrapped Dual Policy Iteration (BDPI), a novel model-free reinforcement-learning algorithm for continuous states and discrete actions, with an actor and several *off-policy* critics. Off-policy critics are compatible with experience replay, ensuring high sample-efficiency, without the need for off-policy corrections. The actor, by slowly imitating the average greedy policy of the critics, leads to high-quality and state-specific exploration, which we compare to Thompson sampling. Because the actor and critics are fully decoupled, BDPI is remarkably stable, and unusually robust to its hyper-parameters. BDPI is significantly more sample-efficient than Bootstrapped DQN, PPO, and ACKTR, on discrete, continuous and pixel-based tasks. Source code: `https://github.com/vub-ai-lab/bdpi`. Appendix: `https://arxiv.org/abs/1903.04193`.

Keywords: reinforcement learning; value iteration; actor-critic

1 Introduction and Related Work

State-of-the-art stochastic actor-critic algorithms, used with discrete actions, all share a common trait: the critic Q^π they learn directly evaluates the actor [23, 37, 47, 26]. Some algorithms allow the agent to execute a policy different from the actor, which the authors refer to as off-policy, but the critic is still on-policy with regards to the actor [18, for instance]. ACER and the off-policy actor-critic [44, 12] use off-policy corrections to learn Q^π from past experiences, DDPG learns its critic with an on-policy SARSA-like algorithm [24], Q-prop [17] uses the actor in the critic learning rule to make it on-policy, and PGQL [28] allows for an off-policy V function, but requires it to be combined with on-policy advantage values. Notable examples of algorithms without an on-policy critic are AlphaGo Zero [38], that replaces the critic with a slow-moving target policy learned with tree search, and the Actor-Mimic [31], that minimizes the cross-entropy between an actor and the Softmax policies of critics (see Section 4.2). The need of most actor-critic algorithms for an on-policy critic makes them incompatible with

state-of-the-art value-based algorithms of the Q-Learning family [3, 20], that are all highly sample-efficient but off-policy. In a *discrete-actions* setting, where off-policy value-based methods can be used, this raises two questions:

1. Can we use *off-policy* value-based algorithms in an actor-critic setting?
2. Would the actor bring anything positive to the agent?

In this paper, we provide a positive answer to these two questions. We introduce Bootstrapped Dual Policy Iteration (BDPI), a novel actor-critic algorithm. Our actor learning rule, inspired by Conservative Policy Iteration (see Sections 2.4 and 3.2), is robust to off-policy critics. Because we lift the requirement for on-policy critics, the full range of value-based methods can now be leveraged by the critic, such as DQN-family algorithms [20], or exploration-focused approaches [3, 9]. To better isolate the sample-efficiency and exploration properties arising from our actor-critic approach, we use in this paper a simple DQN-family critic. We learn several Q-Functions, as suggested by [30], with a novel extension of Q-Learning (see Section 3.1). Unlike other approaches, that use the critics to compute means and variances [27, 11], BDPI uses the information in each individual critic to train the actor. We show that our actor learning rule, combined with several off-policy critics, can be compared to bootstrapped Thompson sampling (Section 3.4).

Our experimental results in Section 4 show that BDPI significantly outperforms state-of-the-art actor-critic *and* critic-only algorithms, such as PPO, ACKTR and Bootstrapped DQN, on a set of discrete, continuous and 3D-rendered tasks. Our ablative study shows that BDPI's actor significantly contributes to its performance and exploration. To the best of our knowledge, this is the first time that, in a discrete-action setting, the benefit of having an actor can be clearly identified. Finally, and perhaps most importantly, BDPI is highly robust to its hyper-parameters, which mitigates the need for endless tuning (see Section 4.5). BDPI's ease of configuration and sample-efficiency are crucial in many real-world settings, where computing power is not the bottleneck, but data collection is.

2 Background

In this section, we introduce and review the various formalisms on which Bootstrapped Dual Policy Iteration builds. We also compare current actor-critic methods with Conservative and Dual Policy Iteration, in Sections 2.3 and 2.4.

2.1 Markov Decision Processes

A discrete-time Markov Decision Process (MDP) [6] with discrete actions is defined by the tuple $\langle S, A, R, T, \gamma \rangle$: a possibly-infinite set S of states; a finite set A of actions; a reward function $R(s_t, a_t, s_{t+1}) \in \mathbb{R}$ returning a scalar reward r_{t+1} for each state transition; a transition function $T(s_{t+1}|s_t, a_t) \in [0, 1]$ determining the dynamics of the environment; and the discount factor $0 \leq \gamma < 1$ defining the importance given by the agent to future rewards.

A stochastic stationary policy $\pi(a_t|s_t) \in [0,1]$ maps each state to a probability distribution over actions. At each time-step, the agent observes s_t, selects $a_t \sim \pi(.|s_t)$, then observes r_{t+1} and s_{t+1}, which produces an $(s_t, a_t, r_{t+1}, s_{t+1})$ *experience* tuple. An optimal policy π^* maximizes the expected cumulative discounted reward $\mathbb{E}_{\pi^*}[\sum_t \gamma^t r_t]$. The goal of the agent is to find π^* based on its experience within the environment, with no *a-priori* knowledge of R and T.

2.2 Q-Learning, Experience Replay and Clipped DQN

Value-based reinforcement learning algorithms, such as Q-Learning [45], use experience tuples and Equation 1 to learn an action-value function Q^*, also called a *critic*, which estimates the expected return for each action in each state when the optimal policy is followed:

$$\begin{aligned} Q_{k+1}(s_t, a_t) &= Q_k(s_t, a_t) + \alpha\delta_{k+1} \\ \delta_{k+1} &= r_{t+1} + \gamma \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a_t) \end{aligned} \tag{1}$$

with $0 < \alpha < 1$ a learning rate. At acting time, the agent selects actions having the largest Q-Value, plus some exploration. To improve sample-efficiency, experience tuples are stored in an *experience buffer*, and are periodically re-sampled for further training using Equation 1 [25]. Before convergence, Q-Learning tends to over-estimate the Q-Values [19], as positive errors are propagated by the max operator of Equation 1. Clipped DQN [14], that we use as the basis of our critic learning rule (Section 3.1), addresses this bias by applying the max operator to the minimum of the predictions of two independent Q-functions, such that positive errors are removed by the minimum operation. Addressing this over-estimation has been shown to increase sample-efficiency and robustness [19].

2.3 Policy Gradient and Actor-Critic Algorithms

Instead of choosing actions according to Q-Values, Policy Gradient methods [46, 40] explicitly learn an *actor* $\pi_\theta(a_t|s_t) \in [0,1]$, parametrized by a weights vector θ, such as the weights of a neural network. The objective of the agent is to maximize the expected cumulative discounted reward $\mathbb{E}_\pi[\sum_t \gamma^t r_t]$, which translates to the minimization of Equation 2 [40]:

$$\mathcal{L}(\pi_\theta) = -\sum_{t=0}^{T} \left\{ \begin{matrix} \mathcal{R}_t \\ Q^{\pi_\theta}(s_t, a_t) \end{matrix} \right\} \log(\pi_\theta(a_t|s_t)) \tag{2}$$

with $a_t \sim \pi_\theta(s_t)$ the action executed at time t, and $\mathcal{R}_t = \sum_{\tau=t}^{T} \gamma^\tau r_\tau$ the Monte-Carlo return from time t onwards. At every training epoch, experiences are used to compute the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ of Equation 2, then the weights of the policy are adjusted by a small step in the opposite direction of the gradient. A second

gradient update requires fresh experiences [40], which makes Policy Gradient quite sample-inefficient. Three approaches have been proposed to increase the sample-efficiency of Policy Gradient: trust regions, that allow larger gradient steps to be taken [36], surrogate losses, that prevent divergence if several gradient steps are taken [37], and stochastic[3] actor-critic methods [4, 23], that replace the Monte-Carlo R_t with an estimation of its expectation, $Q^{\pi_\theta}(s_t, a_t)$, an *on-policy* critic, shown in Equation 2, bottom.

The use of Q^{π_θ}-Values instead of Monte-Carlo returns leads to a gradient of lower variance, and allows actor-critic methods to obtain impressive results on several challenging tasks [44, 15, 26]. However, conventional actor-critic algorithms may not provide any benefits over a cleverly-designed critic-only algorithm, see for example [28], Section 3.3. Actor-critic algorithms also rely on Q^{π_θ} to be accurate for the current actor, even if the actor itself can be distinct from the actual behavior policy of the agent [12, 44, 16]. Failing to ensure this accuracy may cause divergence [23, 40].

2.4 Conservative and Dual Policy Iteration

Approximate Policy Iteration and Dual Policy Iteration are two approaches to Policy Iteration. API repeatedly evaluates a policy π_k, producing an *on-policy* Q^{π_k}, then trains π_{k+1} to be as close as possible to the greedy policy $\Gamma(Q^{\pi_k})$ [21, 35]. Conservative Policy Iteration (CPI) extends API to *slowly* move π towards the greedy policy [33]. Dual Policy Iteration [39] formalizes as CPI several modern reinforcement learning approaches [2, 38], by replacing the greedy function with a *slow-moving* target policy π':

$$
\begin{aligned}
& \Gamma(Q^{\pi_k}) && \text{(API)} \\
\pi_{k+1} \leftarrow\ & (1-\alpha)\pi_k + \alpha\Gamma(Q^{\pi_k}) && \text{(CPI)} \\
& (1-\alpha)\pi_k + \alpha\pi'_k && \text{(DPI)}
\end{aligned}
$$

with $0 < \alpha \leq 1$ a learning rate, set to a small value in Conservative Policy Iteration algorithms (0.01 in our experiments). Among CPI algorithms, Safe Policy Iteration [33] dynamically adjusts the learning rate to ensure (with high probability) a monotonic improvement of the policy, while [41] propose the use of statistical tests to decide whether to update the policy.

While theoretically promising, CPI algorithms present two important limitations: their convergence is difficult to obtain with function approximation [43, 7]; and their update rule and associated set of bounds and proofs depend on Q^{π_k}, an *on-policy* function that would need to be re-computed before every iteration in an on-line setting. As such, CPI algorithms are notoriously difficult to implement, with [33] reporting some of the first empirical results on CPI. Our main contribution, presented in the next section, is inspired by CPI but

[3] Deterministic actor-critic methods are slightly different and outside the scope of this paper.

distinct from it in several key aspects. Our actor learning rule follows the Dual Policy Iteration formalism, with a target policy π' built from off-policy critics (see Section 3.2). The fact that the actor gathers the experiences on which the critics are trained can be compared to the *guidance* that π gives to π' in the DPI formalism [39].

3 Bootstrapped Dual Policy Iteration

Our main contribution, Bootstrapped Dual Policy Iteration (BDPI), consists of two original components. In Section 3.1, we introduce an aggressive off-policy critic, inspired by Bootstrapped DQN and Clipped DQN [30, 14]. In Sections 3.2 to 3.3, we introduce an actor that leads to high-quality exploration, further enhancing sample-efficiency. We detail BDPI's exploration properties in Section 3.4, before empirically validating our results in a diverse set of environments (Section 4). Our implementation of BDPI is available on `https://github.com/vub-ai-lab/bdpi`.

3.1 Aggressive Bootstrapped Clipped DQN

We begin our description of BDPI with the algorithm used to train its critics, Aggressive Bootstrapped Clipped DQN (ABCDQN). Like Bootstrapped DQN [30], ABCDQN consists of $N_c > 1$ critics. Combining ABCDQN with an actor is detailed in Section 3.2. When used without an actor, ABCDQN selects actions by randomly sampling a critic for each episode, then following its greedy function.

Each critic of ABCDQN is trained with an aggressive algorithm loosely inspired by Clipped DQN and Double Q-Learning [14, 19]. Each critic maintains two Q-functions, Q^A and Q^B. Every *training iteration*, Q^A and Q^B are swapped, then Q^A is trained with Equation 3 on a set of experiences sampled from an experience buffer, shared by all the critics. Contrary to Clipped DQN, an on-policy algorithm that uses $V(s_{t+1}) \equiv \min_{l=A,B} Q^l(s_{t+1}, \pi(s_{t+1}))$ as target value, ABCDQN removes the reference to $\pi(s_{t+1})$ and instead uses the following formulas:

$$\begin{aligned} Q^A_{k+1}(s_t, a_t) &= Q^A_k(s_t, a_t) + \alpha\big(r_{t+1} + \gamma V(s_{t+1}) - Q^A_k(s_t, a_t)\big) \\ V(s_{t+1}) &\equiv \min_{l=A,B} Q^l\big(s_{t+1}, \operatorname{argmax}_{a'} Q^A_k(s_{t+1}, a')\big) \end{aligned} \tag{3}$$

We increase the aggressiveness of ABCDQN by performing several *training iterations* per *training epoch*. Every *training epoch*, every critic is updated using a different batch of experiences, for $N_t > 1$ *training iteration*. As mentioned above, a training iteration consists of applying Equation 3 on the critic, which produces Q_{k+1} values, either stored in a tabular critic, or used to optimize the parameters of a parametric critic Q_θ. The parameters minimize $\sum_{(s,a)}(Q_\theta(s,a) - Q_{k+1}(s,a))^2$, using gradient descent for *several* gradient steps.

ABCDQN achieves high sample-efficiency (see Section 4), but its purposefully exaggerated aggressiveness makes it prone to overfitting. We now introduce an actor, that alleviates this problem and leads to high-quality exploration, comparable to Thompson sampling (see Section 3.4).

3.2 Training the Actor with Off-Policy Critics

To improve exploration, and further increase sample-efficiency, we now complement our ABCDQN critic with the second component of BDPI, its actor. The actor π takes inspiration from Conservative Policy Iteration [33], but replaces on-policy estimates of Q^π with our off-policy ABCDQN critics. Every *training epoch*, after every critic i has been updated on its batch of experiences $E_i \subset B$ uniformly sampled from the experience buffer, the actor is sequentially trained towards the greedy policy of all the critics:

$$\pi(s) \leftarrow (1-\lambda)\pi(s) + \lambda\Gamma(Q^{A,i}_{k+1}(s,\cdot)) \qquad \forall\ i, \forall\ s \in E_i \tag{4}$$

with $\lambda = 1 - e^{-\delta}$ the actor learning rate, computed from the maximum allowed KL-divergence δ defining a *trust-region* (see Appendix B), and Γ the greedy function, that returns a policy greedy in $Q^{A,i}$, the Q^A function of the i-th critic. Pseudocode for the complete BDPI algorithm is given in the appendix, and summarized in Algorithm 1.

Contrary to Conservative Policy Iteration algorithms, and because our critics are off-policy, the greedy function is applied on an estimate of Q^*, the optimal Q-function, instead of Q^π. The use of an actor, that slowly imitates approximations of $\Gamma(Q^*) \equiv \pi^*$, leads to an interesting relation between BDPI and Thompson sampling (see Section 3.4). While expressed in the tabular form in Equations 3 and 4, the BDPI update rules produce Q-Values and probability distributions that can directly be used to train any kind of function approximator, on the mean-squared-error loss, and for as many gradient steps as desired. The Policy Distillation literature [34] suggests that implementing the actor and critics with neural networks, with the actor having a smaller architecture than the critic, may lead to good results. Large critics reduce bias [13], and a small policy has been shown to outperform and generalize better than big policies [34]. In this

for *every critic* $i \in [1, N_c]$ **do**
  $E \leftarrow$ N experiences sampled from the buffer;
  for N_t *training iterations* **do**
    Swap $Q^{A,i}$ and $Q^{B,i}$;
    Update $Q^{A,i}$ of critic i on E with Equation 3;
  end
  Update actor on E with Equation 4;
end

Algorithm 1: Learning with Bootstrapped Dual Policy Iteration (summary)

paper, we use actors and critics of the same size, and leave the evaluation of asymmetric architectures for future work.

3.3 BDPI and Conservative Policy Iteration

The standard Conservative Policy Iteration update rule (see Section 2.4) updates the actor π towards $\Gamma(Q^\pi)$, the greedy function according to the Q-Values arising from π. This slow-moving update, and the inter-dependence between π and Q^π, allows several properties to be proven [21], and the optimal policy learning rate α to be determined from Q^π [33]. Because BDPI learns off-policy critics, that can be arbitrarily different from the on-policy Q^π function, the Approximate Safe Policy Iteration framework [33] would infer an "optimal" learning rate of 0. Fortunately, a non-zero learning rate still allows BDPI to learn efficiently. In Section 3.4, we show that the off-policy nature of BDPI's critics makes it approximate Thompson sampling, which CPI's on-policy critics do not do. Our experimental results in Section 4 further illustrate how BDPI allows fast and robust learning, even in difficult-to-explore environments.

3.4 BDPI and Thompson Sampling

In a bandit setting, Thompson sampling [42] is regarded as one of the best ways to balance exploration and exploitation [1, 10]. Thompson sampling consists of maintaining a posterior belief of how likely any given action is optimal, and drawing actions directly from this probability distribution. In a reinforcement-learning setting, Thompson sampling consists of selecting an action a according to $\pi(a|s) \equiv P(a = \text{argmax}_{a'} Q^*(s, a'))$, with Q^* the optimal Q-function.

BDPI learns off-policy critics, that produce estimates of Q^*. Sampling a critic and updating the actor towards its greedy policy is therefore equivalent to sampling a function $Q \sim P(Q = Q^*)$ [30], then updating the actor towards $\Gamma(Q)$, with $\Gamma(Q)(s, a) = \mathbb{1}[a = \text{argmax}_{a'} Q(s, a')]$, and $\mathbb{1}$ the indicator function. Over several updates, and thanks to a small λ learning rate (see Equation 4), the actor learns the expected greedy function of the critics, which (intuitively) folds the indicator function into the sampling of Q, leading to an actor that learns $\pi(a|s) = P(a = \text{argmax}_{a'} Q^*(s, a'))$, the Thompson sampling equation for reinforcement learning.

The use of an explicit actor, instead of directly sampling critics and executing actions as Bootstrapped DQN does [30], positively impacts BDPI's performance (see Section 4). [27] discuss why Bootstrapped DQN, without an actor, leads to a higher regret than their Information Directed Sampling, and propose to add a Distributional RL [5] component to their agent. [29] presents arguments against the use of Distributional RL, and instead combines Bootstrapped DQN with prior functions. In the next section, we show that BDPI largely outperforms Boostrapped DQN, along with PPO and ACKTR, without relying on Distributional RL nor prior functions. We believe that having an explicit actor changes the way the posterior is computed, which may positively influence exploration compared to actor-less approaches.

4 Experiments

To illustrate the properties of BDPI, we compare it to its ablations and a wide range of reinforcement learning algorithms, in four environments with completely different state-spaces and dynamics. Our results demonstrate the high sample-efficiency and exploration quality of BDPI. Moreover, these results are obtained with the same configuration of critics, experience replay and learning rates across environments, which illustrates the ease of configuration of BDPI. In Section 4.5, we carry out further experiments, that demonstrate that BDPI is more robust to its hyper-parameters than other algorithms. This is key to the application of reinforcement learning to real-world settings, where vast hyper-parameter tuning is often infeasible.

4.1 Algorithms

We evaluate the algorithms listed below:

BDPI	*this paper*
ABCDQN, BDPI without an actor	*this paper*
BDPI w/ AM, see Section 4.2	*this paper*
BDQN, Bootstrapped DQN	[30]
PPO	[37]
ACKTR	[47]

Except on *Hallway*,[4] a 3D environment described in the next section, all algorithms use feed-forward neural networks to represent their actor and critic, with one (2 for PPO and ACKTR) hidden layers of 32 neurons (256 on *LunarLander*). The state is one-hot encoded in *FrozenLake*, and directly fed to the network in the other environments. The neural networks are trained with the Adam optimizer [22], using a learning rate of 0.0001 (0.001 for PPO, ACKTR uses its own optimizer with a varying learning rate). Unless specified otherwise, BDPI uses $N_c = 16$ critics, all updated every time-step on a different 256-experiences batch, sampled from the same shared experience buffer, for 4 applications of our ABCDQN update rule. BDPI trains its neural networks for 20 epochs per training iteration, on the mean-squared-error loss (even for the policy).

Hallway being a 3D environment, the algorithms are configured differently. Changes to BDPI are minimal, as they only consist of using the standard DeepMind convolutional layers, a hidden layer of 256 neurons, and optimizing the networks for 1 epoch per training iteration, instead of 20. PPO and ACKTR, however, see much larger changes. They use the DeepMind layers, 16 replicas of the environment (instead of 1), a learning rate of 0.00005, and perform gradient steps every 80 time-steps (per replica, so 1280 time-steps in total). These PPO and ACKTR parameters are recommended by the author of *Hallway*.

[4] `https://github.com/maximecb/gym-miniworld`

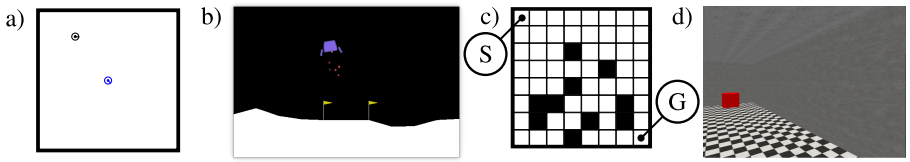

Fig. 1. The four environments. a) *Table*, a large continuous-state environment with a black circular robot and a blue charging station. b) *LunarLander*, a continuous-state task based on the Box2D physics simulator. c) *Frozen Lake*, an 8-by-8 slippery gridworld where black squares represent fatal pits. d) *Hallway*, a 3D pixel-based navigation task.

4.2 BDPI with the Actor-Mimic Loss

To the best of our knowledge, the Actor-Mimic [31] is the only actor-critic algorithm, along with BDPI, that learns critics that are off-policy with regards to the actor. The Actor-Mimic is designed for transfer learning tasks. One critic per task is trained, using the off-policy DQN algorithm. Then, the cross-entropy between the actor and the Softmax policies $S(Q_i)$ of all the critics is minimized, using the (simplified) loss of Equation 5.

$$\mathcal{L}(\pi_\theta) = -\sum_{s \in S, a \in A, i < N} S(Q_i)(a|s) \log(\pi_\theta(a|s)) \tag{5}$$

Applying the Actor-Mimic to a single-task setting is possible. We implemented an agent based on BDPI, that retains its ABCDQN critics, but replaces our actor learning rule of Equation 4 with the Actor-Mimic loss of Equation 5. Because we only change how the actor is trained, and still use our aggressive critics, we ensure the fairest comparison between our actor learning rule and the cross-entropy loss of the Actor-Mimic. In our experiments, the Actor-Mimic loss with Softmax policies fails to learn efficiently, even after extensive hyper-parameter tuning, probably because the Softmax prevents the policy from becoming deterministic in states where this is necessary. We therefore replaced the Softmax with the greedy function, which led to the much better results that we present in Section 4.4.

4.3 Environments

Our evaluation of BDPI takes place in four environments that challenge the algorithms on different aspects of reinforcement learning: exploration with sparse rewards (*Table*), high-dimensional state-spaces (vector *LunarLander*, pixel-based *Hallway*), and high stochasticity (*FrozenLake*).

Table simulates a tiny robot on a large table that has to locate its charging station and dock (see Figure 1a). The table is a 1-by-1 square. The goal is located at $(0.5, 0.5)$, and the robot always starts at $(0.1, 0.1)$, facing away from the goal. A fixed initial position makes exploration more challenging, as the robot

never spawns close to the goal. The robot observes its current (x, y, θ) position and orientation, with $\theta \in [-\pi, \pi]$. Three actions allow the robot to either move forward 0.005 units, or turn left/right 0.1 radians. A reward of 0 is given every time-step. The episode finishes with a reward of -50 if the robot falls off the table, 0 after 200 time-steps, and 100 when the robot successfully docks, that is, its location is $(0.5 \pm 0.05, 0.5 \pm 0.05, \frac{\pi}{4} \pm 0.3)$. The slow speed of the robot and reward sparsity make *Table* more difficult to explore than most Gym tasks [8].

LunarLander is a high-dimensional continuous-state physics-based simulation of a rocket landing on the moon (see Figure 1b). The agent observes the location and velocities of various components of the lander, and has access to four actions: doing nothing, firing the left/right side engines for one time-step, and firing the main engine. The reward signal for this task is quite complicated but informative, as it directly maps the distance between the rocket and the landing pad to a reward, on every time-step. The environment is considered solved when a cumulative reward of 200 or more is achieved per episode [8].

FrozenLake is a 8×8 grid composed of slippery cells, holes, and one goal cell (see Figure 1c). The agent can move in four directions (up, down, left or right), with a probability of $\frac{2}{3}$ of actually performing an action other than intended. The agent starts at the top-left corner of the environment, and has to reach the goal at its bottom-right corner. The episode terminates when the agent reaches the goal, resulting in a reward of +1, or falls into a hole, resulting in no reward.

Hallway is a 3D pixel-based environment, that simulates a camera-based robotic task in the real world. *Hallway* consists of a rectangular room with a target red box, and the agent. The size of the room, location of the goal and initial position of the agent are randomly chosen for each episode. Four discrete actions allow the agent to move forward/backward and turn left/right. Movement is slow, and the amount of movement is stochastic for each time-step. The reward signal is sparse: 0 every time-step, and 1 when the goal is reached. The episode ends with a reward of 0 after 500 time-steps. This sparse reward function heavily stresses the ability of a reinforcement-learning algorithm to train deep convolutional neural networks on small amounts of reward data.

4.4 Results

Figure 2 shows the cumulative reward per episode obtained by various agents in our four environments. These results are averaged across 8 runs per agent, with the shaded regions representing the standard error. The plots compare BDPI to the algorithms detailed in Section 4.1, and display the effect of varying key hyper-parameters of BDPI.

Algorithms BDPI is the most sample-efficient of all the algorithms, and also achieves the highest returns (especially on hard-to-explore *Table* and pixel-based *Hallway*). BDPI with the Actor-Mimic loss matches BDPI with our actor learning rule on *Table*, but fails to learn *LunarLander* and *Hallway*. ABCDQN (BDPI without its actor) fails on *Table*, an environment where exploration is key, and is generally inferior to BDPI. These results show that both having an

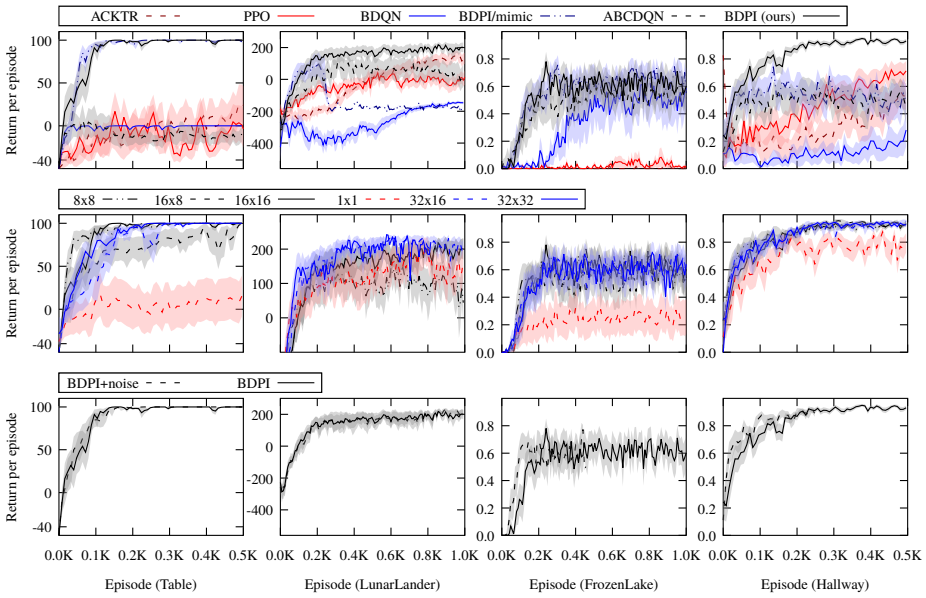

Fig. 2. Results on our four environments. *Top:* BDPI (16 critics, updated for 4 iterations per time-step) outperforms all the other algorithms in every environment. *Middle:* Varying the number of critics and how often they are trained, as long as there are more than one critic, only has minimal impact on BDPI's performance, which demonstrates its robustness. *Bottom:* Adding off-policy noise (see text) does not impact BDPI on any of the environments.

explicit actor, and training it with our update rule of Section 3.2, are necessary to achieve top performance. Bootstrapped DQN is highly sample-efficient on *FrozenLake*, but does not explore well enough on the other environments. PPO and ACKTR, after extensive tuning and with several implementations tested, are not as sample-efficient as BDPI and Bootstrapped DQN, two off-policy algorithms using experience replay. Even with per-environment hyper-parameters, PPO and ACKTR need about 5K episodes to learn *FrozenLake*, and 1K episodes on *Table*. BDPI is the only algorithm that, with a single configuration for all the environments, automatically adjusts to the complexity of a task to achieve maximum sample-efficiency.

Interestingly, PPO and ACKTR do perform well on 3D *Hallway*. We tentatively point out that, due to the prevalence of pixel-based environments in the modern reinforcement-learning literature, current algorithms and hyper-parameters may focus more on the representation learning problem than on the reinforcement learning aspect of tasks. Also note that on *Hallway*, PPO and ACKTR use 16 replicas of the environment (instead of 1 for BDPI, and PPO/ACKTR on the other environments). This setting greatly stabilizes the algorithms, but cannot be applied to real-world physical robots.

Critics Increasing the number of critics leads to smoother learning curves in every environment, at the cost of sample-efficiency in *Table*, where a higher variance in the bootstrap distribution of critics seems to help with exploration. Having only one critic seriously degrades BDPI's performance, and having less than 16 critics is detrimental on *LunarLander*, where the environment dynamics are complex. This indicates that more critics are beneficial in complex environments, but may slightly reduce pure exploration.

Off-Policy noise BDPI's actor learning equations do not refer to any behavior policy or on-policy return, and its critics are learned with a variant of Q-Learning. This hints at BDPI being an off-policy algorithm. We now empirically confirm this intuition. In this experiment, training episodes have, at each time-step, a probability of 0.2 that the agents executes a random action, instead of what the actor wants (0.05 on *Table*, where docking requires precise moves). Testing episodes do not have this noise. The agent learns only from training episodes. Such off-policy noise does not negatively impact BDPI's learning performance. Robustness to off-policy execution is an important property of BDPI for safety-critical tasks with backup policies.

The performance of BDPI, obtained with a single set of hyper-parameters for all the environments[5], demonstrate BDPI's sample-efficiency, high-quality exploration, and strong robustness to hyper-parameters, as rigorously detailed in the next section.

4.5 Robustness to Hyper-Parameters

Hyper-parameters often need to be tweaked depending on the environment. Therefore, it is highly desirable that an algorithm provides good performance even if not optimally configured, as BDPI does. To objectively measure an algorithm's robustness to its hyper-parameters, we draw inspiration from sensitivity analysis. Thousands of runs of the algorithm are performed on randomly-sampled configurations of hyper-parameters, with each configuration evaluated on the total reward obtained over 800 episodes on *LunarLander*. Then, we compute the average absolute difference of total reward between random pairs of configurations, weighted by their distance in configuration space. This measures how much changing hyper-parameters affects performance. The appendix gives more details, and lists the hyper-parameters we consider for each algorithm.

We evaluated numerous algorithms available in the OpenAI baselines. The algorithms, sorted by ascending sensitivity, are DQN with Prioritized ER (930), BDPI (1167), vanilla DQN (1326), A2C (2369), PPO (2452), then ACKTR (5815). Figure 5 in the appendix shows that the apparent robustness of DQN-family algorithms comes from them performing equally badly for every configuration. 35% of BDPI's configurations outperform the best configuration among all the other algorithms.

[5] Only the number of hidden neurons changes between some environments, a trivial change.

5 Conclusion and Future Work

In this paper, we propose Bootstrapped Dual Policy Iteration (BDPI), an algorithm where a bootstrap distribution of aggressively-trained off-policy critics provides an imitation target for an actor. Multiple critics, combined with our actor learning rule, lead to high-quality exploration, comparable to bootstrapped Thompson sampling. Off-policy critics can be learned with any state-of-the-art value-based algorithm, depending on the application domain. BDPI is easy to implement, and remarkably robust to its hyper-parameters. The hyper-parameters we used for the highly-stochastic *FrozenLake* gridworld allowed BDPI to largely outperform the state of the art on three other environments, one of which pixel-based. This, and the availability of BDPI's full source code, makes it one of the first plug-and-play reinforcement-learning algorithm that can easily be applied to new tasks.

While we focus on discrete actions in this paper, the high-quality exploration and robustness to sparse rewards of BDPI lead to encouraging results with discretized continuous action spaces. In Figure 3, we show that Binary Action Search, an approach that allows precise control of continuous actions, at the cost of increased sparsity in the reward function [32], allows BDPI to outperform the Soft Actor-Critic and TD3, three state-of-the-art continuous-actions algorithms. In future work, we will explore and evaluate various discretization approaches, pursuing the goal of applying BDPI to today's complicated continuous-action tasks.

Acknowledgments

The first and second authors are funded by the Science Foundation of Flanders (FWO, Belgium), respectively as 1129319N Aspirant, and 1SA6619N Applied Researcher.

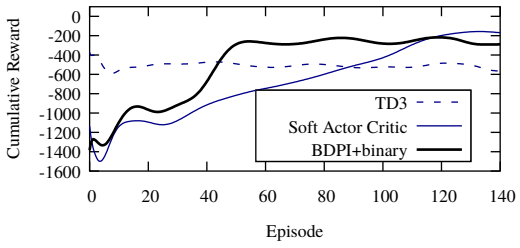

Fig. 3. BDPI adjusted for continuous actions with Binary Action Search [32] is more sample-efficient than TD3 [14, seems to quickly learn to spin] and the Soft Actor-Critic [18] on the Inverted Pendulum task.

Bibliography

[1] Agrawal, S., Goyal, N.: Analysis of thompson sampling for the multi-armed bandit problem. In: Conference on Learning Theory (COLT) (2012)

[2] Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. In: Advances in Neural Information Processing Systems (NIPS). pp. 5366–5376 (2017)

[3] Arjona-Medina, J.A., Gillhofer, M., Widrich, M., Unterthiner, T., Hochreiter, S.: RUDDER: return decomposition for delayed rewards. Arxiv **abs/1806.07857** (2018)

[4] Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Systems, Man, and Cybernetics **13**(5), 834–846 (1983)

[5] Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning. In: International Conference on Machine Learning (ICML). pp. 449–458 (2017)

[6] Bellman, R.: A Markovian decision process. Journal Of Mathematics And Mechanics **6**, 679–684 (1957)

[7] Böhmer, W., Guo, R., Obermayer, K.: Non-deterministic policy improvement stabilizes approximated reinforcement learning. Arxiv **abs/1612.07548** (2016)

[8] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (2016)

[9] Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. arXiv **abs/1810.12894** (2018)

[10] Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. In: Advances in Neural Information Processing Systems (NIPS). pp. 2249–2257 (2011)

[11] Chen, R.Y., Sidor, S., Abbeel, P., Schulman, J.: UCB exploration via q-ensembles. arXiv **abs/1706.01502** (2017)

[12] Degris, T., White, M., Sutton, R.S.: Linear off-policy actor-critic. In: International Conference on Machine Learning, (ICML) (2012)

[13] Fu, J., Kumar, A., Soh, M., Levine, S.: Diagnosing bottlenecks in deep q-learning algorithms. arXiv **abs/1902.10250** (2019)

[14] Fujimoto, S., Hoof, H.V., Meger, D.: Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning (ICML). pp. 1582–1591 (2018)

[15] Gruslys, A., Azar, M.G., Bellemare, M.G., Munos, R.: The reactor: A sample-efficient actor-critic architecture. Arxiv **abs/1704.04651** (2017)

[16] Gu, S., Lillicrap, T., Turner, R.E., Ghahramani, Z., Schölkopf, B., Levine, S.: Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In: Advances in Neural Information Processing Systems (NIPS). pp. 3849–3858 (2017)

[17] Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R.E., Levine, S.: Q-prop: Sample-efficient policy gradient with an off-policy critic. In: International Conference on Learning Representations (ICLR) (2017)
[18] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv **abs/1801.01290** (2018)
[19] van Hasselt, H.: Double Q-Learning. In: Neural Information Processing Systems (NIPS). p. 9 (2010)
[20] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M.G., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. Arxiv **abs/1710.02298** (2017)
[21] Kakade, S., Langford, J.: Approximately optimal approximate reinforcement learning. In: International Conference on Machine Learning (ICML). pp. 267–274 (2002)
[22] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
[23] Konda, V.R., Borkar, V.S.: Actor-Critic-Type Learning Algorithms for Markov Decision Processes. SIAM Journal on Control and Optimization **38**(1), 94–123 (jan 1999)
[24] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv **abs/1509.02971** (2015)
[25] Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning **8**(3-4), 293–321 (1992)
[26] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: International Conference on Machine Learning (ICML). p. 10 (2016)
[27] Nikolov, N., Kirschner, J., Berkenkamp, F., Andreas, K.: Information-directed exploration for deep reinforcement learning. In: International Conference on Learning Representations (ICLR) (2019), in preparation
[28] O'Donoghue, B., Munos, R., Kavukcuoglu, K., Mnih, V.: PGQ: Combining policy gradient and Q-learning. In: International Conference on Learning Representations (ICLR). p. 15 (2017)
[29] Osband, I., Aslanides, J., Cassirer, A.: Randomized prior functions for deep reinforcement learning. arXiv **abs/1806.03335** (2018)
[30] Osband, I., Blundell, C., Pritzel, A., Van Roy, B.: Deep exploration via bootstrapped DQN. In: Advances in Neural Information Processing Systems (NIPS) (2016)
[31] Parisotto, E., Ba, J., Salakhutdinov, R.: Actor-mimic: Deep multitask and transfer reinforcement learning. In: International Conference on Learning Representations (ICLR) (2016)
[32] Pazis, J., Lagoudakis, M.G.: Binary action search for learning continuous-action control policies. In: International Conference on Machine Learning (ICML). pp. 793–800. ACM (2009)

[33] Pirotta, M., Restelli, M., Pecorino, A., Calandriello, D.: Safe policy iteration. In: Proceedings of the 30th International Conference on Machine Learning, (ICML). pp. 307–315 (2013)
[34] Rusu, A.A., Colmenarejo, S.G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., Hadsell, R.: Policy distillation. arXiv **abs/1511.06295** (2015)
[35] Scherrer, B.: Approximate policy iteration schemes: A comparison. In: Proceedings of the 31th International Conference on Machine Learning (ICML). pp. 1314–1322 (2014)
[36] Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning (ICML) (2015)
[37] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. Arxiv **abs/1707.06347** (2017)
[38] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354 (2017)
[39] Sun, W., Gordon, G.J., Boots, B., Bagnell, J.A.: Dual policy iteration. Arxiv **abs/1805.10755** (2018)
[40] Sutton, R., McAllester, D., Singh, S., Mansour, Y.: Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Neural Information Processing Systems (NIPS). p. 7 (2000)
[41] Thomas, P.S., Theocharous, G., Ghavamzadeh, M.: High confidence policy improvement. In: International Conference on Machine Learning (ICML). pp. 2380–2388 (2015)
[42] Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**(3/4), 285–294 (1933)
[43] Wagner, P.: A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In: Advances in Neural Information Processing Systems (NIPS). pp. 2573–2581 (2011)
[44] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample Efficient Actor-Critic with Experience Replay. Tech. rep. (2016)
[45] Watkins, C., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (1992)
[46] Williams, R.J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning **8**(3), 229–256 (1992)
[47] Wu, Y., Mansimov, E., Grosse, R.B., Liao, S., Ba, J.: Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In: Advances in neural information processing systems (NIPS). pp. 5279–5288 (2017)