Off-Policy and Off-Actor Actor-Critic with Bootstrapped Dual Policy Iteration

Denis Steckelmacher,[1] Hélène Plisnier,[1] Diederik M. Roijers[2] and **Ann Nowé[1]**
[1] Vrije Universiteit Brussel
[2] Vrije Universiteit Amsterdam

Abstract

In actor-critic algorithms, a critic Q^π helps to improve an actor π by evaluating its behavior. When function approximation is used, achieving a fruitful coupling between the actor and the critic is challenging, as the accuracy of the critic regarding π is crucial for convergence. We believe that this explains why, in discrete-actions settings, current actor-critic algorithms do not outperform critic-only ones. In this paper, we propose Bootstrapped Dual Policy Iteration (BDPI), a novel model-free on-line reinforcement learning algorithm for discrete action-spaces. BDPI is *off-policy* and *off-actor*: the agent may execute a behavior policy distinct from the actor, and the critic learns Q^* instead of Q^π. BDPI learns several off-policy and off-actor critics, each of which leads to a greedy policy. Samples from this distribution of greedy policies are used as imitation targets by the actor. BDPI has two main benefits over current actor-critic algorithms: the use of the greedy function, instead of actual Q-Values, makes BDPI more resilient to approximation errors in the critics; and the bootstrap distribution of critics leads to an effective exploration strategy, similar to Thompson sampling. Moreover, BDPI is able to aggressively exploit, as the actor learns from an estimate of Q^* rather than some Q^π, that may include an arbitrary exploration strategy. We empirically demonstrate BDPI's superior sample-efficiency and stability compared to a representative selection of state-of-the-art value-based and actor-critic approaches, in three challenging environments.

1 Introduction

In discrete-action settings, the two main families of model-free reinforcement-learning algorithms are actor-critic and critic-only. Critic-only algorithms learn a value function, from which they infer an implicit policy, and are built on increasingly complicated extensions of Q-Learning (Schaul et al., 2015; Anschel et al., 2017; Arjona-Medina et al., 2018). Actor-critic algorithms learn an explicit actor, in addition to the value function. Actor-critic algorithms offer better convergence guarantees and increased stability compared to critic-only algorithms (Konda & Borkar, 1999), but current actor-critic algorithms match but do not outperform critic-only ones in practice (Wang et al., 2016; Gruslys et al., 2017; Schulman et al., 2017).

In Section 2.3, we review current actor-critic methods, and discuss their potential limiting factors. Then, in Section 2.4, we review Conservative Policy Iteration algorithms (Scherrer, 2014, CPI), an alternative form of actor-critic that does not use Policy Gradient. CPI algorithms offer strong convergence guarantees and regret bounds (Kakade & Langford, 2002; Pirotta et al., 2013), but, like actor-critic algorithms, require an *on-actor* critic. Such a critic, learned on-policy with regards to the actor, makes the implementation of sample-efficient experience replay limited and challenging (Wang et al., 2016). Moreover, any approximation error in the critic largely impacts the convergence guarantees of actor-critic and CPI algorithms (Pirotta et al., 2013). Instead of an on-actor critic, AlphaGo Zero (Silver et al., 2017) uses a slow-moving target policy. Sun et al. (2018) call this algorithmic property Dual Policy Iteration.

In this paper, we propose Bootstrapped Dual Policy Iteration (BDPI), a novel algorithm that addresses the main limitations of current actor-critic and CPI algorithms. Our contribution is two-fold. First, BDPI learns several off-policy and *off-actor* critics with an algorithm inspired from Clipped DQN (Fujimoto et al., 2018, see Section 3.1). The greedy policies of these critics approximate a posterior of what the agent thinks the optimal policy is (Osband & Van Roy, 2015). Second, samples of that distribution are used as target policies for the actor, using an actor learning rule inspired from Conservative Policy Iteration (see Section 3.2). Learning an explicit actor ensures stable learning and efficient exploration, that we compare to Thompson sampling in Section 3.6. Our experimental results (Section 4) demonstrate the sample-efficiency and stability of BDPI, and show that it outperforms a representative set of state-of-the-art model-free reinforcement-learning algorithms.

2 Background

In this section, we introduce and review the various formalisms on which Bootstrapped Dual Policy Iteration builds. We also discuss the shortcomings of current actor-critic algorithms (Section 2.3), and contrast them to Conservative Policy Iteration and Dual Policy Iteration.

2.1 Markov Decision Processes

A discrete-time Markov Decision Process (MDP) (Bellman, 1957) 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 $R(s_t, a_t, s_{t+1}) \in R$ 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 timestep, 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*. An optimal policy π^* maximizes the expected cumulative discounted reward $E_{\pi^*}[\sum_t \gamma^t r_t]$. The goal of the agent is to find π^* based on its experiences 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 (Watkins & Dayan, 1992), use experience tuples and Equation 1 to learn an action-value function Q_π, also called a *critic*, which estimates the expected return for each action in each state:

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

with $0 < \alpha < 1$ a learning rate. At acting time, the agent selects actions having the largest Q-Value, following some exploration strategy. To improve sample-efficiency, experience tuples are stored in an *experience buffer*, and are periodically re-sampled for further training using Equation 1 (Lin, 1992). When function approximation is used, Q-Learning tends to over-estimate the Q-Values (van Hasselt, 2010), as positive errors are propagated by the $\max$ of Equation 1. Clipped DQN (Fujimoto et al., 2018), that we use as the basis of our critic learning rule (Section 3.1), addresses this bias by applying the max to the minimum of the predictions of two independent Q-functions, such that positive errors are removed by the minimum operation.

2.3 Policy Gradient and Actor-Critic Algorithms

Instead of choosing actions according to Q-Values, Policy Gradient methods (Williams, 1992; Sutton et al., 2000) explicitly learn an *actor* policy $\pi_\theta(a_t|s_t) \in [0, 1]$, 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 $E_\pi[\sum_t \gamma^t r_t]$, which translates to the minimization of the following equation (Sutton et al., 2000):

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

with $a_t \sim \pi_\theta(s_t)$ the action executed at time t, $\mathcal{R}_t = \sum_{\tau=t}^{T} \gamma^\tau r_\tau$ the Monte-Carlo return at time t, and $r_\tau = R(s_\tau, a_\tau, s_{\tau+1})$. 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 one small step in the opposite direction of the gradient. A second gradient update requires fresh experiences (Sutton et al., 2000), 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 (Schulman et al., 2015), surrogate losses, that prevent divergence if several gradient steps are taken (Schulman et al., 2017), and actor-critic methods (Barto et al., 1983; Konda & Borkar, 1999), that learn an *on-policy* critic in addition to the actor, and use its $Q^{\pi_\theta}(s_t, a_t)$ prediction instead of R_t in Equation 2.

Actor-Critic methods allow the actor to use Q-Values instead of Monte-Carlo returns, which leads to a gradient of lower variance. Actor-Critic methods achieve impressive results on several challenging tasks (Wang et al., 2016; Gruslys et al., 2017; Mnih et al., 2016; Schulman et al., 2017). However, conventional actor-critic algorithms may not provide any benefits over a cleverly-designed critic-only algorithm, see for example O'Donoghue et al. (2017), Equation 4. 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 (Degris et al., 2012; Wang et al., 2016; Gu et al., 2017). Maintaining a close and healthy relationship between the actor and the critic requires careful design, as discussed by Konda & Borkar (1999) and Sutton et al. (2000).

2.4 Conservative Policy Iteration 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 $\Gamma(Q^{\pi_k})$ (Kakade & Langford, 2002; Scherrer, 2014). Conservative Policy Iteration (CPI) is an extension of API that slowly moves π towards the greedy policy, which increases the stability of the algorithm (Pirotta et al., 2013). Dual Policy Iteration (Sun et al., 2018, DPI), a formalization of recent advances in reinforcement-learning, follows the same general approach, but replaces the greedy function with a *slow-moving* target policy π'. The target policy π'_k is learned with expensive Monte-Carlo methods, guided by π_k, while the fast-moving policy π_{k+1} imitates π'_k using Conservative Policy Iteration (Sun et al., 2018, Equation 6):

$$\pi_{k+1} = (1 - \alpha)\pi_k + \alpha \left\{ \begin{array}{c} \Gamma(Q^{\pi_k}) \\ \pi' \end{array} \right\} \begin{array}{l} \textit{(CPI)} \\ \textit{(DPI)} \end{array} \quad (3)$$

with $0 < \alpha \leq 1$ a learning rate, set to a small value in Conservative Policy Iteration algorithms (0.05 in our experiments). Among CPI algorithms, Safe Policy Iteration (Pirotta et al., 2013) dynamically adjusts the learning rate to ensure (with high probability) a monotonic improvement of the policy, while Thomas et al. (2015) 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 (Wagner, 2011; Böhmer et al., 2016); and their update rule and entire 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 Pirotta et al. (2013) proposing some of the first empirical results on CPI. Our main contribution, presented in the next sub-section, is inspired by CPI but distinct from it in several key aspects. Its actor learning rule (Section 3.2) follows the Dual Policy Iteration formalism, with an easy-to-compute π' target policy, that does not rely on simulations, but instead uses off-policy critics. Its critic learning rule (Section 3.1) is off-policy and uses experience replay, which increases sample-efficiency and stability. The fact that the actor influences the experiences gathered by the agent can be compared to the *guidance* that π gives to π' in DPI algorithms (Sun et al., 2018).

2.5 Dealing with Uncertainty

Measuring the uncertainty of an agent is crucial to ensure proper exploration. Many approaches to uncertainty estimation exist, each with their own slightly different definition of uncertainty, and are divided in two big families: changing the learning algorithm or function approximator to produce uncertainty measures; or using several actors or critics, without interfering with low-level aspects of the agent. Fundamental ways of measuring uncertainty include Distributional Reinforcement Learning (Bellemare et al., 2017), that models the distribution of Q-Values, Bayes-by-Backprop (Blundell et al., 2015), that allows a neural network to express a distribution over outputs, or the addition of stochastic noise in the parameters of a neural network (Plappert et al., 2017). These methods assume that a neural network is used, and fundamentally change how it is trained.

The second approach to modeling uncertainty consists of using several actors or critics, leading to a bootstrap distribution whose statistics give sound measures of uncertainty (Efron & Tibshirani, 1994). This approach does not need the learning algorithm to be fundamentally changed, which makes it more easily applicable. For this reason, this is the approach we use in this paper. It is also used by A3C, that deploys several actors in replicas of the environment (Mnih et al., 2016), Bootstrapped DQN, that uses only one environment and bootstraps critics instead of actors (Osband et al., 2016), and even modern implementations of Proximal Policy Optimization (Schulman et al., 2017).

3 Bootstrapped Dual Policy Iteration

Our main contribution, Bootstrapped Dual Policy Iteration (BDPI), consists of two original components. In Section 3.1, we propose a value-based algorithm, Aggressive Bootstrapped Clipped DQN (ABCDQN), inspired from Bootstrapped DQN and Clipped DQN (Osband et al., 2016; Fujimoto et al., 2018) but much more aggressive than either of those. By learning several critics, ABCDQN produces a bootstrap distribution for $P(a = \text{argmax}_{a'} Q^*(s, a'))$ (Efron & Tibshirani, 1994). Because that bootstrap distribution has a high variance, due to the necessarily limited number of critics it learns, we introduce an actor in Section 3.2. The actor learns the expected greedy policy of the critics, with equations comparable to Dual Policy Iteration, which leads to the complete, stable and sample-efficient BDPI algorithm.

3.1 Aggressive Bootstrapped Clipped DQN

First, we describe the steps that allow to build ABCDQN, the algorithm that produces BDPI's critics. Like Double Q-Learning (van Hasselt, 2010), ABCDQN maintains two Q-functions per critic, Q^A and Q^B. Every *training iteration*, Q^A and Q^B are swapped, then Q^A is trained using the Q-Learning rule (see Equation 1), on a set of experiences sampled from an experience buffer. Building on Double Q-Learning, Clipped DQN (Fujimoto et al., 2018), which should have been called Clipped SARSA, uses $V(s_{t+1}) \equiv \min_{l=A,B} Q^l(s_{t+1}, \pi(s_{t+1}))$ as target value of the Q-Learning equation. The minimum prevents prediction errors from propagating through the max, and allows nice bounds on the error on Q-Values to be computed. Unlike Clipped DQN, ABCDQN is *off-actor*, and learns the optimal Q-function Q^* instead of some Q^π. As such, we propose a new target value,[1] that does not depend on the actor π:

$$V(s_{t+1}) \equiv \min_{l=A,B} Q^l(s_{t+1}, \text{argmax}_{a'} Q^A(s_{t+1}, a')) \quad (4)$$

Bootstrapped DQN (Osband et al., 2016) consists of maintaining N_c critics, each one having its own experience buffer. The agent follows the greedy policy of a single critic, randomly sampled for each episode among the N_c critics. At every time-step, the experience obtained from the environment is added to the experience buffer of a random subset of critics. Every *training epoch*, a critic is chosen randomly and one *training iteration* of Clipped DQN is performed on it. Aggressive Bootstrapped Clipped DQN consists of performing $N_t > 1$ training iterations on the critic, every training epoch, which aggressively updates it towards Q^*. This produces stronger estimates of what the agent thinks the optimal Q-function is, leads to better policies (see Section 4), but also increases overfitting. The actor of BDPI follows this aggressive critic, restoring exploration and approximating Thompson sampling (see Section 3.6).

3.2 Conservative Policy Iteration with an Off-Policy and Off-Actor Critic

The Aggressive Bootstrapped Clipped DQN algorithm described in the previous section leads to impressive results in our experiments (see Section 4), but the variance of its bootstrap distribution of critics is sometimes too high. To address this problem, our complete Bootstrapped Dual Policy Iteration algorithm introduces an actor π, trained using a variant of Conservative Policy Iteration (Pirotta et al., 2013) tailored to off-policy critics. At every training epoch, after one of the critics i has been updated on a sample $E \subset B_i$ of experiences from its experience buffer, the same sample is used to update the actor towards the greedy policy of the critic:

$$\pi_{k+1}(s) = (1 - \lambda)\pi_k(s) + \lambda\Gamma(Q^{A,i}_{k+1}(s, \cdot)) \quad \forall s \in E \quad (5)$$

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 $Q^{A,i}$. Pseudocode for the complete BDPI

[1] Equation 4 empirically outperforms alternative target values.

algorithm is given in Appendix A. Contrary to Conservative Policy Iteration algorithms, the greedy function is applied on a randomly-sampled estimate of Q^*, the optimal Q-function, instead of Q^π, the on-policy Q-function. The impact of this difference is discussed in Section 3.3. The use of an actor, that slowly imitates the greedy functions, leads to an interesting relation between BDPI and Thompson sampling (see Section 3.6). While expressed in the tabular form in the above equation, the BDPI update rules produce Q-Values and probability distributions that can directly be used to train any kind of parametric or non-parametric function approximator, on the mean-squared-error loss, and for as many gradient steps as desired.

3.3 BDPI and Actor-Critic Algorithms

Even if BDPI maintains an actor and several critics, it is different from conventional actor-critic algorithms in two fundamental ways: its actor update rule does not use Policy Gradient, and the critic is off-policy and off-actor.

Actor Update Contrary to conventional actor-critic algorithms based on Policy Gradient (see Equation 2), neither the action a_t taken at time t, nor the on-policy return $\mathcal{R}_t$, appear in the actor update rule, which makes it off-policy, and ensures that old experiences being replayed do not drag the agent towards old low-quality policies. Moreover, relying on the greedy policy of a critic, instead of its absolute Q-Values (as Equation 2 does), makes BDPI more robust to approximation errors than conventional actor-critic algorithms (Bellemare et al., 2016, discuss why large differences in Q-Values are desirable).

Critic Update While conventional actor-critic algorithms use simple critics, that approximate the expected return under π (Mnih et al., 2016; Schulman et al., 2017) or learn Q^π from off-policy experiences (Wang et al., 2016), BDPI learns off-policy and off-actor critics that approximate Q^* instead of Q^π. This aspect of BDPI also makes it distinct from conventional CPI algorithms, as detailed in the next sub-section, and simplify its convergence analysis (see Section 3.5).

3.4 BDPI and Conservative Policy Iteration

The standard Conservative Policy Iteration update rule (see Equation 3) 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 (Kakade & Langford, 2002), and the optimal policy learning rate α to be determined from Q^π (Pirotta et al., 2013).

Because BDPI learns off-policy critics, that represent a bootstrap distribution of Q^*, its update rule makes the actor π follow a target policy $\Gamma(Q^*)$ that can be arbitrarily different from π. In the Approximate Safe Policy Iteration framework of Pirotta et al. (2013), this amounts to an unbounded error between Q^* and the exact Q^π function considered in the proof, which leads to an "optimal" learning rate of 0. Fortunately, BDPI's critics still allow stable and efficient learning, using a non-zero learning rate, as we discuss in the next sub-section. We also observed that BDPI exhibits desirable exploration behaviors, such as state-dependent, almost novelty-based exploration (see Appendix C), and higher exploration when the critics are uncertain. Uncertainty in the value function leads to critics that have different greedy functions (Osband et al., 2016), which in turn prevents the actor from learning a single deterministic policy.

3.5 Convergence

Because the actor and the critic of BDPI are decoupled, the convergence analyses of the conventional Conservative Policy Iteration literature do not apply to BDPI. However, decoupling the actor and the critic make their analysis much simpler: if the critic learns the optimal Q^* function for any behavior policy, and if the actor is able to fully imitate $\Gamma(Q^*) \equiv \pi^*$, then the actor learns the optimal policy.

Convergence of the critic with an arbitrary behavior policy has been shown, in the tabular setting, for one-step Q-Learning (Watkins & Dayan, 1992) and Retrace (Munos et al., 2016). The very recent SBEED algorithm (Dai et al., 2018) extends the convergence guarantees to non-linear function approximation, such as neural networks. Clipped DQN, that we use in this paper, does not enjoy the same convergence properties as SBEED, but our empirical results of Section 4 show that complicated and highly-advanced critic learning methods are not required to achieve stable and efficient learning. The decoupled nature, modularity and generality of BDPI allow Clipped DQN to be replaced with SBEED in a setting where proven convergence is necessary.

The actor is slowly updated towards $\Gamma(Q)$, with Q converging towards Q^*. Because the actor update rule of Equation 5 can be seen as a simple running average, π will eventually converge to $\Gamma(Q^*) \equiv \pi^*$ once the critic has converged. This shows that if the critic converges to Q^*, the actor converges to π^*. Our experimental results of Section 4 demonstrate the convergence of BDPI in a practical setting, with BDPI being the only algorithm able to learn a non-degenerate policy on our simulated robotic task.

3.6 BDPI and Thompson Sampling

In a bandit setting, Thompson sampling (Thompson, 1933) is regarded as one of the best ways to balance exploration and exploitation (Agrawal & Goyal, 2012; Chapelle & Li, 2011). Thompson sampling defines the policy as $\pi(a) \equiv P[a = \text{argmax}_{a'} R(a')]$, the probability, according to the current belief of the agent, that a is the optimal action. In a reinforcement-learning setting, Thompson sampling consists of selecting an action a according to:

$$\pi(a|s) \equiv P(a = \text{argmax}_{a'} Q^*(s, a')) \tag{6}$$

with Q^* the optimal Q-function. BDPI learns its critics with Aggressive Bootstrapped Clipped DQN, that produces strong estimates of Q^* (due to the aggressiveness, see Section 3.1). Sampling a critic and updating the actor towards its greedy policy is therefore equivalent to sampling $Q \sim P(Q = Q^*)$ (Osband et al., 2016), 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 that returns

1 when a condition is true, 0 otherwise. Intuitively, this $\mathbb{1}$ can be folded into the sampling of Q to produce the Thompson sampling equation (6). More precisely, over several updates, and thanks to a small λ learning rate (see Equation 5), the actor learns the expected greedy function of its critics:

$$\begin{aligned}\pi(a|s) &= E_{Q\sim P(Q=Q^*)}[\mathbb{1}(a = \text{argmax}_{a'}\, Q(s,a'))] \\ &= \int_Q \mathbb{1}(a = \text{argmax}_{a'}\, Q(s,a'))P(Q=Q^*)dQ \\ &= \int_Q P(a = \text{argmax}_{a'}\, Q(s,a'), Q=Q^*) \\ &\qquad \hookrightarrow P(Q=Q^*)dQ \\ &= P(a = \text{argmax}_{a'}\, Q^*(s,a'))\end{aligned}$$

The above equations show that BDPI's actor learns to approximate Thompson sampling.

3.7 BDPI and Pursuit

BDPI, and Conservative Policy Iteration algorithms in general, are closely related to Pursuit algorithms. Pursuit, originally designed for bandit settings (Berry & Fristedt, 1985), has been first applied to a reinforcement learning context in REINFORCE (Williams, 1992). Inherently actor-critic, Pursuit algorithms maintain a critic $Q(s,a)$ and an actor $\pi(a|s)$. At each time-step t, the critic is updated using a value-based learning algorithm, and the actor is updated by *pursuing* a set of actions (Narendra & Thathachar, 1989), using Equation 7:

$$\pi_{k+1}(s) = (1-\alpha)\pi_k(s) + \alpha\frac{\pi'_{k+1}(s)}{\|\pi'_{k+1}(s)\| + \epsilon}, \qquad (7)$$

with π_0 a uniform probability distribution, $\mathbf{0}$ the vector of all zeros, $0 < \alpha < 1$ a learning rate, and ϵ positive but very close to zero. The main difference between Conservative Policy Iteration and Pursuit is that Equation 7 uses a vector $\pi'_{k+1}(s)$ where there may be *several* ones, instead of the greedy function Γ that associates exactly one action to every state. Several Pursuit schemes exist, and change how $\pi'_{k+1}(s)$ is produced:

$$\pi'_{t+1}(a|s) = \mathbb{1}(a = a^* \wedge a = a_t), \qquad \text{(RI)}$$

$$\pi'_{t+1}(a|s) = \mathbb{1}(a = a^*), \qquad \text{(RP)}$$

$$\pi'_{t+1}(a|s) = \mathbb{1}(a = a^* \vee Q(s,a) > Q(s,a_t)), \qquad \text{(GP)}$$

with $a^* = \text{argmax}_{a'}\, Q(s,a')$ the greedy action. Reward-Penalty (RP) pursues the greedy action (Thathachar & Sastry, 1986). Reward-Inaction (RI) pursues the current action if it is greedy, and does nothing otherwise (Shapiro & Narendra, 1969). Generalized Pursuit (GP) pursues the greedy action, in addition to every action strictly better than the current one (Agache & Oommen, 2002). Conservative Policy Iteration, by updating the actor towards the greedy policy of the critic, is equivalent to Reward-Penalty. In Section 4.3, we evaluate how various Pursuit schemes influence BDPI's actor learning, and show that Reward-Penalty indeed provides the best results, probably because a_t does not appear in its formula, which decouples the actor update from the actual behavior of the agent.

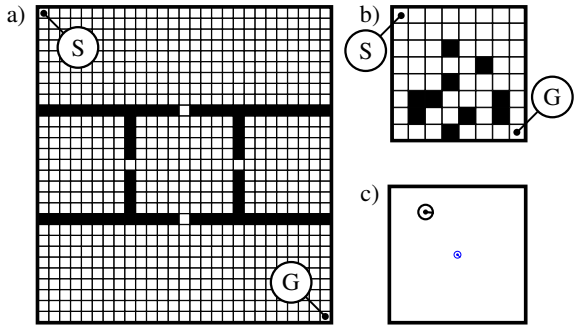

Figure 1: The three environments. a) Five rooms, a 29-by-27 gridworld where black squares represent walls. b) *Frozen Lake*, an 8-by-8 gridworld where black squares represent fatal pits. c) *Table*, a large continuous-state environment where the black circle represents a robot, and the blue one a charging station.

4 Experiments

In order to assess the properties of BDPI, we compare it to various state-of-the-art reinforcement learning algorithms, in three environments that each test for different desirable aspects of a general reinforcement learning algorithm. Our main results are discussed in Section 4.3, and illustrate the high sample-efficiency, exploration quality and stability of BDPI. In our most challenging environment, BDPI learns the optimal policy, while the other algorithms do not even find the goal.

4.1 Algorithms

We compare BDPI to various state-of-the-art reinforcement learning algorithms, on the three environments described in Section 4.2. We compare our BDPI algorithm to Aggressive Bootstrapped Clipped DQN (ABCDQN, BDPI without an actor), Bootstrapped DQN (Osband et al., 2016, BDQN), and Proximal Policy Optimization (Schulman et al., 2017, PPO). We also evaluated ACER and A3C (Wang et al., 2016; Mnih et al., 2016), conventional actor-critic algorithms available in the OpenAI baselines, but obtained much worse results. As such, we omitted these results in our plots, for increased readability.

All algorithms use feed-forward neural networks to represent their actor and critic. In discrete environments, they take as input the one-hot encoding of the current state. In continuous environments, the raw observations are directly fed to the network. Every algorithm trains its networks with the Adam optimizer (Kingma & Ba, 2014), using a learning rate of 0.0001, and has been configured in a best effort to produce good results (parameters in Appendix D).

BDPI Configuration and Variants In all the experiments, BDPI uses $N_c = 16$ critics, all updated on a different 512-experiences batch after every time-step. BDPI trains its neural networks for 20 epochs per training iteration, and adds every experience to the buffer of every critic. The critics are different due to their random weight initialization,

and the fact that they each see a different batch of experiences every training iteration.

To fully understand the contributions of each component of BDPI, we evaluate it with different Pursuit schemes (see Section 3.7), vary the number C of critics updated per training epoch (among 16 critics), and vary the number N_t of training iterations performed on each of these critics. In the figures, we denote as $C \times N_t$ a run of BDPI with 16 critics, with C critics updated per training epoch, and each of these critic seeing N_t Clipped DQN iterations.

4.2 Environments

Our evaluation of BDPI takes place in three environments that stress different properties of reinforcement learning algorithms. *Frozen Lake* (8x8) is a highly-stochastic environment, available in the OpenAI Gym (Brockman et al., 2016), and assesses the robustness of algorithms to high stochasticity. *Five Rooms* is our own large and difficult-to-explore environment, inspired from the well-known Four Rooms environment (Precup et al., 1998) but much larger. *Table* represents a high-level robotic task, with continuous states and complex dynamics, that exerts the stability of neural networks.

Frozen Lake is a 8×8 grid composed of slippery cells, holes, and one goal cell (see Figure 1b). 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. The best policy available for this environment obtains a cumulative reward of a bit less than 0.9 on average.

Five Rooms is a 29 cells high and 27 cells wide grid, divided by walls into five rooms connected by thin doors (see Figure 1a, compare with the 13×13 grid of the conventional Four Rooms environment). The agent has access to four actions, that allow it to deterministically move up, down, left or right. When the agent tries to move into a wall, nothing happens. The agent starts at the top-left corner of the environment, and has to reach the goal located in the bottom-right corner of the bottom room. The agent receives a reward of -1 per time-step, and the episode terminates with a reward of -50 after 500 time-steps, or with a reward of +100 if the agent reaches the goal. The absence of stochasticity in this environment, and the sparsity of the reward signal, makes exploration difficult. The optimal policy takes the shortest path to the goal, and obtains a cumulative reward of 47.

Table consists of a simulated round robot on a table, able to move forward or to change direction in small steps, that has to locate its charging station and dock on it, without falling from the table (see Figure 1c). The table is big, the robot is slow, the charging station is small, and the robot has to dock in the right orientation for the task to be completed. This makes *Table* more challenging and difficult to explore than many Gym tasks, possibly some Atari games included, despite its simple state space. The table itself is a 1-by-1 square. The goal is located at $(0.5, 0.5)$, and the robot always starts at $(0.1, 0.1)$. A random initial position makes the task easier by providing exploration for free, so we chose a fixed initial position far from the goal. The robot observes its current (x, y, θ) position and orientation, with the orientation being expressed in radians in the $[-\pi, \pi]$ interval. Three actions allow the robot to either move forward 0.005 units (in the orientation it faces), turn left 0.1 radians, or turn 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 $(0.5 \pm 0.05, 0.5 \pm 0.05, \frac{\pi}{4} \pm 0.3)$.

4.3 Results

Figure 2 shows the cumulative reward per episode obtained by various agents in our three environments. These results are averaged across 8 runs per agent, with the light regions representing the standard deviation. The top-most plots compare BDPI (with 16 critics, updated every time-step for 4 Clipped DQN iterations) to the algorithms detailed in Section 4.1. The middle row compares the three main Pursuit schemes used as BDPI's actor learning rule, and the bottom plots display the effect of varying the number of critics updated per training epoch, and the number of Clipped DQN iterations performed on them.

Algorithms BDPI is the most sample-efficient of all the algorithms, on every task, and is the only algorithm able to learn on *Table*. Bootstrapped DQN is less sample-efficient than BDPI, but thanks to its high-quality exploration, almost always learns the optimal policies (except on *Table*). PPO is less sample-efficient than the other algorithms, and does not manage to learn good policies in every run. In *Five Rooms*, where episodes are capped at 500 time-steps, setting PPO's batch size to 500 is crucial for it to learn anything. BDPI does not require such environment-specific parameter tuning, yet outperforms every other algorithm both in sample-efficiency and stability.

Pursuit Variants Generalized Pursuit (Agache & Oommen, 2002, GP) and Reward-Penalty (Thathachar & Sastry, 1986, RP) perform comparably, with Reward-Penalty being slightly more stable and achieving slightly higher returns on *Frozen Lake* and *Table*. Reward-Penalty corresponds to pursuing the greedy policy of the critic, and is therefore equivalent to Conservative Policy Iteration, while Generalized Pursuit is more recent and complicated, but does not decouple the actor update rule from the actual behavior of the agent (see Section 3.7).

Hyper-parameters Updating all the critics at every training epoch (black lines), instead of only one (blue lines), leads to increased sample-efficiency and stability. Each critic is trained on a different sample of 512 experiences (out of 20000 in the experience buffer), which ensures their diversity. Training critics for several iterations, instead of just once (the aggressive part of ABCDQN), also appears to be critical for stability, with the $C \times 4$ runs consistently learning better policies than the $C \times 1$ runs, in addition to being more sample-efficient.

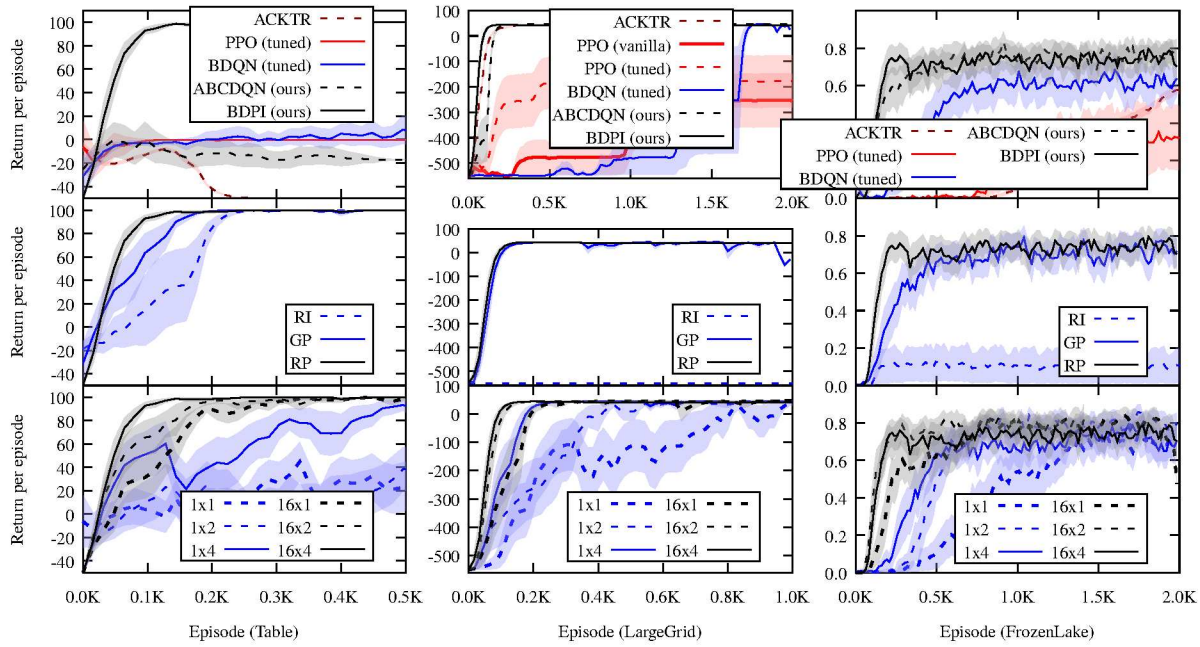

Figure 2: Results on *Table* (left), *Five Rooms* (center) and *Frozen Lake* (right). *Top:* BDPI outperforms ABCDQN (BDPI without an actor), Boostrapped DQN and PPO. *Middle:* Reward-Penalty outperforms other Pursuit schemes. *Bottom:* Updating all the critics, for 4 Clipped DQN iterations each (16×4), provides the best results.

Off-Policy noise To assess the off-policy nature of BDPI, we evaluated it with off-policy noise. Training episodes have, at each time-step, a probability of 0.2 that the agents executes a random action, instead of what its actor wants. Testing episodes do not have that noise. Only experiences from training episodes are added to the experience buffer, and testing episodes are used to produce learning curves. The resulting learning curves are indistinguishable from the ones of BDPI in Figure 2, and thus omitted in the figures. On *Table*, the noise has to be decreased to 0.05, because docking requires a precise sequence of moves.

These results demonstrate the sample-efficiency, quality of exploration and stability of BDPI; show that its actor is necessary for top performance, especially on *Table*; and that BDPI is robust to off-policy execution.

5 Conclusion

In this paper, we proposed Bootstrapped Dual Policy Iteration (BDPI), an algorithm where a bootstrap distribution of aggressively-trained critics provides an imitation target for an actor. Contrary to conventional actor-critic and Conservative Policy Iteration algorithms, the critics are learned using an off-policy and off-actor algorithm, inspired by Clipped DQN. This leads to high-quality exploration and stability, which in turns allows for a high sample-efficiency. BDPI is easy to implement and compatible with any kind of actor and critic representation. We empirically showed that BDPI outperforms state-of-the-art algorithms such PPO and Bootstrapped DQN.

References

Agache, Mariana and Oommen, B. John. Generalized pursuit learning schemes: New families of continuous and discretized learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(6):738–749, 2002.

Agrawal, Shipra and Goyal, Navin. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory (COLT)*, 2012.

Anschel, Oron, Baram, Nir, and Shimkin, Nahum. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 176–185, 2017.

Arjona-Medina, Jose A., Gillhofer, Michael, Widrich, Michael, Unterthiner, Thomas, and Hochreiter, Sepp. RUDDER: return decomposition for delayed rewards. *Arxiv*, abs/1806.07857, 2018.

Barto, Andrew G., Sutton, Richard S., and Anderson, Charles W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

Bellemare, Marc G, Ostrovski, Georg, Guez, Arthur, Thomas, Philip S, and Munos, Rémi. Increasing the Action Gap: New Operators for Reinforcement Learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1476–1483, 2016.

Bellemare, Marc G., Dabney, Will, and Munos, Rémi. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 449–458, 2017.

Bellman, Richard. A Markovian decision process. *Journal Of Mathematics And Mechanics*, 6:679–684, 1957.

Berry, Donald and Fristedt, Bert. *Bandit Problems: Sequential Allocation of Experiments*. Springer, 1985.

Blundell, Charles, Cornebise, Julien, Kavukcuoglu, Koray, and

Wierstra, Daan. Weight uncertainty in neural networks. *Arxiv*, abs/1505.05424, 2015.

Böhmer, Wendelin, Guo, Rong, and Obermayer, Klaus. Non-deterministic policy improvement stabilizes approximated reinforcement learning. *Arxiv*, abs/1612.07548, 2016.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. OpenAI Gym, 2016.

Chapelle, Olivier and Li, Lihong. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2249–2257, 2011.

Dai, Bo, Shaw, Albert, Li, Lihong, Xiao, Lin, He, Niao, Liu, Zhen, Chen, Jianshu, and Song, Le. Sbeed: Convergent reinforcement learning with nonlinear function approximation. In *International Conference on Machine Learning (ICML)*, pp. 1133–1142, 2018.

Degris, Thomas, White, Martha, and Sutton, Richard S. Linear off-policy actor-critic. In *International Conference on Machine Learning, (ICML)*, 2012.

Efron, Bradley and Tibshirani, Robert J. *An introduction to the bootstrap*. CRC press, 1994.

Fujimoto, Scott, Hoof, Herke Van, and Meger, David. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pp. 1582–1591, 2018.

Gruslys, Audrunas, Azar, Mohammad Gheshlaghi, Bellemare, Marc G., and Munos, Rémi. The reactor: A sample-efficient actor-critic architecture. *Arxiv*, abs/1704.04651, 2017.

Gu, Shixiang, Lillicrap, Tim, Turner, Richard E., Ghahramani, Zoubin, Schölkopf, Bernhard, and Levine, Sergey. 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.

Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 267–274, 2002.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Konda, Vijaymohan R. and Borkar, Vivek S. Actor-Critic-Type Learning Algorithms for Markov Decision Processes. *SIAM Journal on Control and Optimization*, 38(1):94–123, jan 1999.

Lin, Long J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321, 1992.

Mnih, Volodymyr, Badia, Adrià Puigdomènech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, pp. 10, 2016.

Munos, Rémi, Stepleton, Thomas, Harutyunyan, Anna, and Bellemare, Marc G. Safe and efficient off-policy reinforcement learning. In *Neural Information Processing Systems (NIPS)*, 2016.

Narendra, Kumpati S. and Thathachar, Mandayam A. L. *Learning automata - an introduction*. Prentice Hall, 1989.

O'Donoghue, Brendan, Munos, Remi, Kavukcuoglu, Koray, and Mnih, Volodymyr. PGQ: Combining policy gradient and Q-learning. In *International Conference on Learning Representations (ICLR)*, pp. 15, 2017.

Osband, Ian and Van Roy, Benjamin. Bootstrapped thompson sampling and deep exploration. *arXiv*, abs/1507.00300, 2015.

Osband, Ian, Blundell, Charles, Pritzel, Alexander, and Van Roy, Benjamin. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Pirotta, Matteo, Restelli, Marcello, Pecorino, Alessio, and Calandriello, Daniele. Safe policy iteration. In *Proceedings of the 30th International Conference on Machine Learning, (ICML)*, pp. 307–315, 2013.

Plappert, Matthias, Houthooft, Rein, Dhariwal, Prafulla, Sidor, Szymon, Chen, Richard Y., Chen, Xi, Asfour, Tamim, Abbeel, Pieter, and Andrychowicz, Marcin. Parameter space noise for exploration. *Arxiv*, abs/1706.01905, 2017.

Precup, Doina, Sutton, Richard S., and Singh, Satinder P. Theoretical results on reinforcement learning with temporally abstract options. In *European Conference on Machine Learning (ECML)*, pp. 382–393, 1998.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized Experience Replay. In *International Conference on Learning Representations (ICLR)*, 2015.

Scherrer, Bruno. Approximate policy iteration schemes: A comparison. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pp. 1314–1322, 2014.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael I., and Moritz, Philipp. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec, and Klimov, Oleg. Proximal policy optimization algorithms. *Arxiv*, abs/1707.06347, 2017.

Shapiro, Joseph and Narendra, Kumpati S. Use of stochastic automata for parameter self-optimization with multimodal performance criteria. *IEEE Trans. Systems Science and Cybernetics*, 5(4):352–360, 1969.

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Sun, Wen, Gordon, Geoffrey J., Boots, Byron, and Bagnell, J. Andrew. Dual policy iteration. *Arxiv*, abs/1805.10755, 2018.

Sutton, Richard, McAllester, D, Singh, Satinder, and Mansour, Yishay. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems (NIPS)*, pp. 7, 2000.

Thathachar, Mandayam AL and Sastry, P Shanthi. Estimator algorithms for learning automata. In *Platinum Jubilee Conference on Systems and Signal Processing*, 1986.

Thomas, Philip S., Theocharous, Georgios, and Ghavamzadeh, Mohammad. High confidence policy improvement. In *International Conference on Machine Learning (ICML)*, pp. 2380–2388, 2015.

Thompson, William 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.

van Hasselt, Hado. Double Q-Learning. In *Neural Information Processing Systems (NIPS)*, pp. 9, 2010.

Wagner, Paul. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2573–2581, 2011.

Wang, Ziyu, Bapst, Victor, Heess, Nicolas, Mnih, Volodymyr, Munos, Rémi, Kavukcuoglu, Koray, and de Freitas, Nando. Sample Efficient Actor-Critic with Experience Replay. Technical report, 2016.

Watkins, Christopher and Dayan, Peter. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Williams, Ronald J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3):229–256, 1992.

Appendix A. Bootstrapped Dual Policy Iteration Pseudocode

The following pseudocode provides a complete description of the BDPI algorithm. In order to keep our notations simple and general, the pseudocode is given for the tabular setting, and does not refer to any parameter for the actor and critics. An actual implementation of BDPI, such as the one we used in our experiments, uses the equations below to produce batches of state-action or state-value pairs. A function approximator, such as a neural network, is then trained on these batches, minimizing the mean-squared-error loss, for several gradient steps.

Algorithm 1 Bootstrapped Dual Policy Iteration

Require: A policy π
Require: N_c critics. $Q^{A,i}$ and $Q^{B,i}$ are the two Clipped DQN networks of critic i.
  procedure BDPI
    for $t \in [1, T]$ **do**
      ACT
      if t a multiple of K **then**
        LEARN
      end if
    end for
  end procedure
  procedure ACT
    Observe s_t
    $a_t \sim \pi(s_t)$
    Execute a_t, observe r_{t+1} and s_{t+1}
    Add $(s_t, a_t, r_{t+1}, s_{t+1})$ to the experience buffer of a random subset of critics
  end procedure
  procedure LEARN
    for every critic $i \in [1, N_c]$ (in random order) **do** ▷ Bootstrapped DQN
      Sample a batch B of N experiences from the i-th experience buffer
      for N_t iterations **do** ▷ Aggressive BDQN
        for all $(s_t, a_t, r_{t+1}, s_{t+1}) \in B$ **do** ▷ Clipped DQN
          $\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \min_{l=A,B} Q^{l,i}(s_{t+1}, \operatorname{argmax}_{a'} Q^{A,i}(s_{t+1}, a'))$
        end for
        Train $Q^{A,i}$ towards $\hat{Q}$ with learning rate α
        Swap $Q^{A,i}$ and $Q^{B,i}$
      end for
      $\pi \leftarrow (1 - \lambda)\pi + \lambda\Gamma(Q^{A,i})$ ▷ CPI with an off-policy critic (DPI)
    end for
  end procedure

Appendix B. The CPI Learning Rate Implements a Trust-Region

A trust-region, successfully used in a reinforcement-learning algorithm by Schulman et al. (2015), is a constrain on the Kullback-Leibler divergence between a policy π_k and an updated

policy π_{k+1}. In BDPI, we want to find a policy learning rate λ such that $D_{KL}(\pi_k||\pi_{k+1}) \leq \delta$, with δ the *trust-region*.

While a trust-region is expressed in terms of the *KL-divergence*, Conservative Policy Iteration algorithms, the family of algorithms to which BDPI belongs, naturally implement a bound on the *total variation* between π and π_{k+1}:

$$\begin{aligned} \pi_{k+1} &= (1-\lambda)\pi_k + \lambda\pi' && \text{see Equations 3 and 5 in the paper} \\ D_{TV}(\pi_{k+1}(s)||\pi_k(s)) &= \sum_a |\pi_{k+1}(a|s) - \pi_k(a|s)| && (7) \\ &\leq 2\lambda \end{aligned}$$

The total variation is maximum when π', the target policy, and π_k, both have an action selected with a probability of 1, and the action is not the same. In CPI algorithms, the target policy is a greedy policy, that selects one action with a probability of one. The condition can therefore be slightly simplified: the total variation is maximized if π_k assigns a probability of 1 to an action that is not the greedy one. In this case, the total variation is 2λ (2 elements of the sum of (7) are equal to λ).

The Pinsker inequality (Pinsker, 1960) provides a lower bound on the KL-divergence based on the total variation. The inverse problem, upper-bounding the KL-divergence based on the total variation, is known as the Reverse Pinsker Inequality. It allows to implement a trust-region, as $D_{KL} \leq f(D_{TV})$ and $D_{TV} \leq 2\lambda$, with $f(D_{TV})$ a function applied to the total variation so that the reverse Pinsker inequality holds. Upper-bounding the KL-divergence to some δ then amounts to upper-bounding $f(D_{TV}) \leq \delta$, which translates to $\lambda \leq \frac{1}{2} f^{-1}(\delta)$.

The main problem is finding f^{-1}. The reverse Pinsker inequality is still an open problem, with increasingly tighter but complicated bounds being proposed (Sason, 2015). A tight bound is important to allow a large learning rate, but the currently-proposed bounds are almost impossible to inverse in a way that produces a tractable f^{-1} function. We therefore propose our own bound, designed specifically for a CPI algorithm, slightly less tight than state-of-the-art bounds, but trivial to inverse.

If we consider two actions, we can produce a policy $\pi_k(s) = \{0, 1\}$ and a greedy target policy $\pi'(s) = \{1, 0\}$. The updated policy $\pi_{k+1} = (1-\lambda)\pi_k + \lambda\pi'$ is, for state s, $\pi_{k+1}(s) = \{\lambda, 1-\lambda\}$. The KL-divergence between π_k and π_{k+1} is:

$$\begin{aligned} D_{KL}(\pi_k||\pi_{k+1}) &= 1 \log \frac{1}{1-\lambda} + 0 \log \frac{0}{\lambda} \\ &= \log \frac{1}{1-\lambda} \end{aligned} \tag{8}$$

if we assume that $\lim_{x \to 0} x \log x = 0$. Based on the reverse Pinsker inequality, we assume that if the two policies used above are greedy in different actions, and therefore have a maximal total variation, then their KL-divergence is also maximal. We use this result to introduce a trust region:

$$D_{KL}(\pi_k||\pi_{k+1}) \leq \delta \qquad \text{trust region}$$

$$\begin{aligned}\log \frac{1}{1-\lambda} &\leq \delta \\ \frac{1}{1-\lambda} &\leq e^{\delta} \\ \lambda &\leq 1 - e^{-\delta}\end{aligned}$$

Interestingly, for small values of δ, as they should be in a practical implementation of BDPI, $1 - e^{-\delta} \approx \delta$. The trust-region is therefore implemented by choosing $\lambda = \delta$, which is much simpler than the line-search method proposed by Schulman et al. (2015).

Appendix C. Environments

Our evaluation of BDPI takes place in three environments, that each assess different properties of reinforcement learning algorithms. Frozen Lake is a highly-stochastic environment, available in the OpenAI Gym (Brockman et al., 2016, we use the large 8×8 variant of it), and assesses the robustness of algorithms to high stochasticity. Five Rooms is our own large and difficult-to-explore environment, inspired from the well-known Four Rooms environment (Precup et al., 1998) but much larger, that assesses the quality of exploration of an algorithm. Table represents a high-level robotic task, with continuous states and complex dynamics, that exerts the stability of neural networks.

Five Rooms is a 29 cells high and 27 cells wide grid, divided by walls into five rooms connected by thin doors (see Figure 1, a, compare with the 13×13 grid of the conventional Four Rooms environment). The agent has access to four actions, that allow it to deterministically move up, down, left or right. When the agent tries to move against a wall, nothing happens. The absence of stochasticity in this environment makes exploration difficult, as the agent cannot wait to eventually drift out of a bad situation. The agent starts at the top-left corner of the environment, and has to reach the goal located in the bottom-right corner of the bottom room. The agent must pass through two thin doors, and must correctly navigate towards the bottom room when visiting the central corridor. Combined with the sheer size of the environment, this makes exploration exceptionally challenging. The agent receives a reward of -1 per time-step, and the episode terminates with a reward of -50 after 500 time-steps, or with a reward of $+100$ if the agent reaches the goal. The optimal policy takes the shortest path to the goal, and obtains a cumulative reward of 46.

Frozen Lake is a 8×8 grid composed of slippery cells, holes, and one goal cell (see Figure 1, b). 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. The best policy available for this environment obtains a cumulative reward of a bit less than 0.9 on average.

Table consists of a wheeled round robot on a table, able to move forward or to change direction in small steps, that has to locate its charging station and dock on it, without falling from the table (see Figure 1, c). The table is big, the robot is slow, the charging station is small, and the robot has to dock in the right orientation for the task to be completed. This

makes Table more challenging and difficult to explore than many Gym tasks, several Atari games included, without requiring pixel-level observations. The table itself is a 1-by-1 square. The goal is located at $(0.5, 0.5)$, and the robot always start at $(0.1, 0.1)$ (a random initial position makes the task easier by providing exploration for free, so we choose a fixed initial position far from the goal). The robot observes its current (x, y, θ) position and orientation, with the orientation being expressed in radians in the $[-\pi, \pi]$ interval. Three actions allow the robot to either move forward 0.005 units (along its current orientation), turn left 0.1 radians, or turn 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, and 100 when the robot successfully docks. Episodes also automatically finish after 2000 time-steps. The robot successfully docks when its location is $(0.5 \pm 0.05, 0.5 \pm 0.05, \frac{\pi}{4} \pm 0.3)$, or $(0.5 \pm 5\%, 0.5 \pm 5\%, \frac{\pi}{4} \pm 4.7\%)$ if tolerances are expressed in percentages of the range their value take.

Appendix D. State-Dependent Exploration

BDPI, by training the actor on the expected policy arising from the critics, removes an important component of Bootstrapped DQN: explicit deep exploration. Deep exploration consists of performing a sequence of directed exploration steps, instead of exploring in a random direction at each time-step (Osband et al., 2016). Bootstrapped DQN achieves deep exploration by greedily following a single critic, sampled at random, for an entire *episode*. By training the actor towards a randomly-selected critic at every *time-step*, BDPI changes its exploration behavior. Figure 3 shows that the entropy of the policy is still lower in some parts of the environment, leading to increased exploitation at some times, and more exploration later on. Our experimental results demonstrate, in our difficult-to-explore environment, that BDPI still outperforms Bootstrapped DQN and our extensions of it (see Figure 2, compare BDPI, ABCDQN – BDPI without its actor –, and tuned Bootstrapped DQN, samples every episode instead of every time-step).

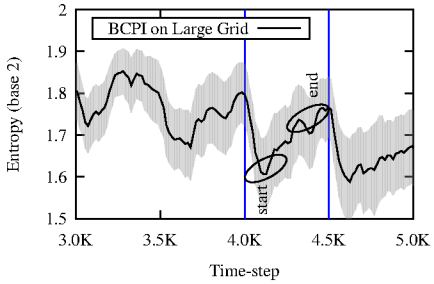

Figure 3: Entropy of the policy per time-step, on the Large Grid environment (running average and standard deviation of 8 runs). At early stages of learning (as shown in the figure), the agent does not yet manage to reach the goal. Episodes are therefore truncated after 500 time-steps. The entropy of the policy is lower in already-explored parts of the environment, at the beginning of episodes, and becomes significantly larger later in the episode, when new regions of the environment are discovered.

	PPO	PPO (tuned)	BDQN	BDQN (tuned)	ABCDQN	BDPI
Discount factor γ	0.99		0.99		0.99	
Replay buffer size	0		20K		20K	
Experiences/batch	256	500	512		512	
Training epoch every K time-steps	256	500	1		1	
Policy loss	PPO		–		–	MSE
Trust region δ	–		–		–	0.001
Entropy regularization	0.01		–		–	0
Critic count N_c	0		1		16	
Critic sampling frequency	–		episode		–	
Critic learning rate α	–		0.2		0.2	
Critic training iterations N_t	–		1		4	
Gradient steps/batch	4		1	20	20	
Learning rate	0.001		0.0001		0.0001	
Hidden layers	1		1		1	
Hidden neurons	100		32		32	
Activation function	tanh		tanh		tanh	

Table 1: Hyper-parameter of the various algorithms we experimentally evaluate. All the parameters are kept constant across runs and environments.

Appendix E. Experimental Setup

All the algorithms evaluated in the two environments described above use feed-forward neural networks to represent their actor(s) and critic(s). They take as input the one-hot encoding of the current state, and are trained with the Adam optimizer (Kingma and Ba, 2014), using a learning rate of 0.0001 (0.001 for PPO, as it gave better results). We configured each algorithm following the recommendations in their respective papers, and further tuned some parameters to the environments we use. These parameters are given Table 1. They are kept as similar as possible across algorithms, and constant across our two environments, to evaluate the generality of the algorithms.