European Workshop on Reinforcement Learning 14 (2018) October 2018, Lille, France.

Stable, Practical and On-line Bootstrapped Dual Policy Iteration

Denis Steckelmacher DSTECKEL@AI.VUB.AC.BE
Hélène Plisnier HPLISNIE@AI.VUB.AC.BE
Vrije Universiteit Brussel, Belgium

Diederik M. Roijers D.M.ROIJERS@VU.NL
Vrije Universiteit Amsterdam, The Netherlands

Ann Nowé ANOWE@COMO.VUB.AC.BE
Vrije Universiteit Brussel, Belgium

Abstract

We consider on-line model-free reinforcement learning with discrete actions, and focus on sample-efficiency and exploration quality. In this setting, value-based methods of the Q-Learning family achieve state-of-the-art results, while actor-critic algorithms, that learn an explicit actor policy in addition to their value function, do not achieve empirical results matching their theoretical promises. The majority of actor-critic algorithms combine an on-policy critic with an actor learned with Policy Gradient. In this paper, we propose an alternative to these two components. We base our work on Conservative Policy Iteration (CPI), leading to a new non-parametric actor learning rule, and Dual Policy Iteration (DPI), that motivates our use of aggressive off-policy critics. Our empirical results demonstrate a sample-efficiency and robustness superior to state-of-the-art value-based and actor-critic approaches, in three challenging environments.

Keywords: Actor-Critic, Conservative Policy Iteration, Bootstrapped DQN, Thompson Sampling

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 and 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 and 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 and 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 3.5. 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 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*. 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 and 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:

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

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):

$$\mathcal{L}(\pi_\theta) = -\sum_{t=0}^{T} \left\{ \begin{array}{c} \mathcal{R}_t \\ Q^{\pi_\theta}(s_t, a_t) \end{array} \right\} \log(\pi_\theta(a_t|s_t)) \qquad \begin{array}{l} \triangleright \text{ actor-only} \\ \triangleright \text{ actor-critic} \end{array} \tag{2}$$

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 and 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 and 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 and 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(Anthony et al., 2017; Silver et al., 2017), 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):

$$\pi_{k+1} = (1-\alpha)\pi_k + \alpha \left\{ \begin{array}{c} \Gamma(Q^{\pi_k}) \\ \pi' \end{array} \right\} \begin{array}{l} \triangleright \text{ conservative policy iteration} \\ \triangleright \text{ dual policy iteration} \end{array} \quad (3)$$

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).

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 and 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:

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

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 3.5).

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:

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

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 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 3.5). 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.

1. Equation 4 empirically outperforms alternative target values.

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

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 and 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 shown in our experiments. We also observed that BDPI exhibits desirable exploration behaviors, such as state-dependent, almost novelty-based exploration (see Appendix D), 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 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 and Goyal, 2012; Chapelle and 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:

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

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

4. Experiments

In order to assess the practical performance of BDPI, we evaluate it on three challenging tasks, depicted in Figure 1: FrozenLake8x8 from the OpenAI Gym (Brockman et al., 2016), Five Rooms, a 29-by-27 gridworld, much larger than the common Four Rooms (Precup et al., 1998), and Table, a continuous-state environment where a robot has to dock on its docking station. Frozen Lake is highly stochastic. Five Rooms, due to its thin doors and deterministic dynamics, and Table, due to the slow speed of the robot and small size of the docking station,

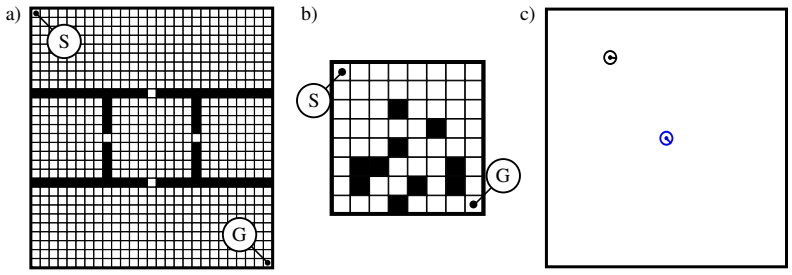

Figure 1: The three environments. a) Five rooms, a 29-by-27 gridworld where black squares represent walls. b) FrozenLake, an 8-by-8 gridworld where black squares represent fatal pits. c) Table, a large continuous-state environment where the black circle represents a robot, and the blue one a charging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

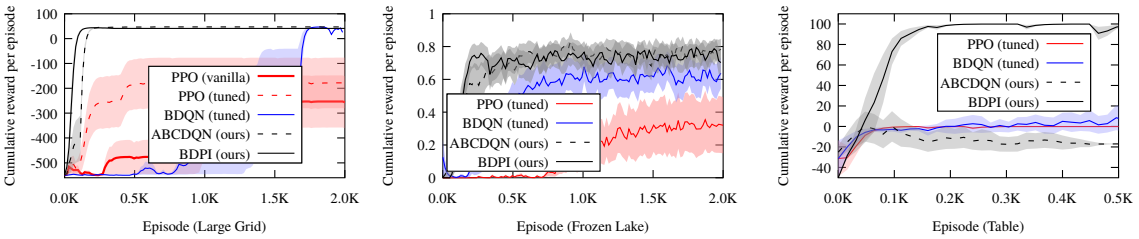

Figure 2: BDPI achieves higher sample-efficiency and stability than other algorithms. Tuning the number of gradient steps performed by BDQN, and the number of time-steps per batch for PPO, is crucial to their performance. ABCDQN and BDPI (ours) perform well on all environments with the same set of hyper-parameters.

are exceptionally difficult to explore. All the environments are fully described in Appendix C. We evaluate BDPI (which we present in this paper), 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 (Wang et al., 2016), a conventional actor-critic algorithm, available in the OpenAI baselines, but obtained disappointing results. As such, we omitted these results in our plots, for increased readability.

Every algorithm is run 8 times in each environment, to produce standard deviation regions in Figure 2. All the agents use feed-forward neural networks, with a single hidden layer of 32 neurons (100 for PPO) and the tanh activation function. The networks take as input the one-hot encoding of the current state. Every algorithm has been configured in a best effort to produce good results, with all the hyper-parameters detailed in Appendix E. BDPI uses $N_c = 16$ critics, all updated on a different 512-experiences sample after every time-step, 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 sample of experiences (512 out of 20000) when they are trained. Figure 2 shows that BDPI outperforms every other algorithm in our three environments, both in sample-efficiency and the quality of the learned policy.

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 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.

Acknowledgments

The first author is "Aspirant" with the Science Foundation of Flanders (FWO, Belgium).

References

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

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

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5366–5376, 2017.

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

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

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

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

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

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

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

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

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

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

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

Shixiang Gu, Tim Lillicrap, Richard E. Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3849–3858, 2017.

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

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

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

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

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

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

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

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

Mark S Pinsker. Information and information stability of random variables and processes. 1960.

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

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

Igal Sason. On reverse pinsker inequalities. *Arxiv*, abs/1503.07118, 2015.

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

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

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

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

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

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

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

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

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

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

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

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

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

Ronald J. Williams. 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
          $\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \min_{l=A,B} Q^{l,i}(s_{t+1}, \text{argmax}_{a'} Q^{A,i}(s_{t+1}, a'))$
        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).

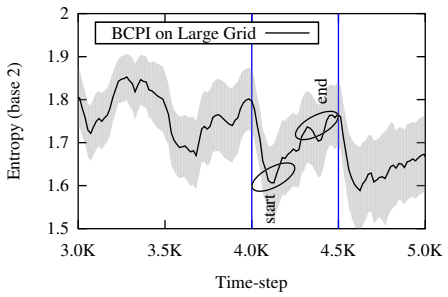

Figure 3: Entropy of the policy per time-step, on the Large Grid environment (running average and standard deviation of 8 runs). At early stages of learning (as shown in the figure), the agent does not yet manage to reach the goal. Episodes are therefore truncated after 500 time-steps. The entropy of 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.