SPIBB-DQN: Safe Batch Reinforcement Learning with Function Approximation

Romain Laroche*
Microsoft Research Montréal
romain.laroche@microsoft.com

Rémi Tachet des Combes*
Microsoft Research Montréal
remi.tachet@microsoft.com

Abstract

We consider Safe Policy Improvement (SPI) in Batch Reinforcement Learning (Batch RL): from a fixed dataset and without direct access to the true environment, train a policy that is guaranteed to perform at least as well as the baseline policy used to collect the data. Our contribution is a model-free version of the SPI with Baseline Bootstrapping (SPIBB) algorithm, called SPIBB-DQN, which consists in applying the Bellman update only in state-action pairs that have been sufficiently sampled in the batch. In low-visited parts of the environment, the trained policy reproduces the baseline. We show its benefits on a navigation task and on CartPole. SPIBB-DQN is, to the best of our knowledge, the first RL algorithm relying on a neural network representation able to train efficiently and reliably from batch data, without any interaction with the environment.

Keywords: Batch Reinforcement Learning, Safe Reinforcement Learning, DQN, Policy Improvement

1 Introduction

Most real-world Reinforcement Learning agents [17, RL] are to be deployed simultaneously on numerous independent devices and cannot be patched quickly. In other practical applications, such as crop management or clinical tests, the outcome of a treatment can only be assessed after several years. Consequently, a bad update could be in effect for a long time, potentially hurting the user's trust and/or causing irreversible damages. Devising safe algorithms with guarantees on the policy performance is a key challenge of modern RL that needs to be tackled before any wide-scale adoption.

Batch RL is an existing approach to such offline settings and consists in training a policy on a fixed set of observations without access to the true environment [10]. It should not be mistaken with the multi-batch setting where the learner trains successive policies from small batches of interactions with the environment [5, 11]. Current Batch RL algorithms are however either unsafe or too costly computationally to be applied to real-world applications.

In this paper, we focus on Safe Policy Improvement (SPI). SPI consists in safely improving a baseline policy from a batch of data. We develop a model-free version of SPI with Baseline Bootstrapping [12, 16, SPIBB]. SPIBB bootstraps the trained policy with the baseline in the state-action pair transitions that were not probed enough in the dataset. Similarly to [14], it assumes access to the baseline. Such a scenario is typically encountered when a policy is trained in a simulator and then run in its real environment, for instance in Transfer RL [19]; or when a system is designed with expert knowledge and then optimized, e.g. in dialogue applications [20, 8].

In Section 2, we recall the basics of SPIBB and implement a model-free version of the algorithm, SPIBB-DQN. SPIBB-DQN relies on pseudo-counts of the state-actions pairs, which can either be handcrafted or approximated using density models [1], and allows to apply SPIBB algorithms to tasks requiring a neural network representation. In Section 3, we apply SPIBB-DQN to a continuous navigation task of our design and to CartPole. SPIBB-DQN is, to the best of our knowledge, the first RL algorithm relying on a neural network representation able to train efficiently and reliably from batch data, without any interaction with the environment [5]. Section 4 concludes the paper and suggests possible future research directions.

2 SPI with Baseline Bootstrapping

Background: An MDP is denoted by $M = \langle \mathcal{X}, \mathcal{A}, R, P, \gamma \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{A}$ is the action space, $R(x,a) \in [-R_{max}, R_{max}]$ is the bounded stochastic reward function, $P(\cdot|x,a)$ is the transition distribution, and $\gamma \in [0,1)$ is the discount factor. The true environment is modelled as an unknown finite MDP $M^* = \langle \mathcal{X}, \mathcal{A}, R^*, P^*, \gamma \rangle$. $\Pi = \{\pi : \mathcal{X} \to \Delta_{\mathcal{A}}\}$ is the set of stochastic policies, where $\Delta_{\mathcal{A}}$ denotes the set of probability distributions over $\mathcal{A}$. The state and state-action value functions are respectively denoted by $V_M^\pi(x)$ and $Q_M^\pi(x,a)$. We define the performance of a policy by its expected return, starting from the initial state x_0: $\rho(\pi, M) = V_M^\pi(x_0)$. Given a policy subset $\Pi' \subseteq \Pi$, a policy π' is said to be Π'-optimal for an MDP M when its performance is maximal in Π': $\rho(\pi', M) = \max_{\pi \in \Pi'} \rho(\pi, M)$.

In this paper, we focus on the batch RL setting where the algorithm does its best at learning a policy from a fixed set of experience. Given a dataset of transitions $\mathcal{D} = \langle x_j, a_j, r_j, x'_j \rangle_{j \in [\![1,N]\!]}$ collected by following a baseline policy π_b, we denote by $\widehat{M} = \langle \mathcal{X}, \mathcal{A}, \widehat{R}, \widehat{P}, \gamma \rangle$ the Maximum Likelihood Estimation (MLE) MDP of the environment. $\widehat{R}$ is the reward mean and $\widehat{P}$ the transition statistics observed in the dataset. Vanilla batch RL looks for the optimal policy in $\widehat{M}$. This policy may be found indifferently using dynamic programming on the explicitly modelled MDP $\widehat{M}$, Q-learning with experience replay until convergence [17], or Fitted-Q Iteration with a one-hot vector representation of the state space [6].

As we shall see below (a fact also observed in [12]), Vanilla batch RL performs very poorly on stochastic environments because it learns from all the transitions in the dataset, including the rarely sampled ones for which the uncertainty is large and which can thus be misleading. This is not too problematic in an online setting, as the policy privileging a poor action in a given state will soon perform it and immediately correct its estimates. In a batch setting however, the bad policy might be in action for a while before its next update, leading to very poor performance on an extended period of time. A solution to this issue is to act pessimistically when the uncertainty is high: this flip side of *optimism in the face of uncertainty* [18] can be achieved by penalizing actions rarely observed in the dataset as e.g. in RaMDP [14]. An alternative option consists in bootstrapping on the baseline, a technique we now present.

SPIBB methodology: SPIBB essentially reformulates the percentile criterion [4]. It consists in optimizing the policy with respect to its performance in the MDP estimate $\widehat{M}$, while guaranteeing it to be ζ-approximately at least as good as π_b in an admissible MDP set Ξ which contains the true MDP M^* with high probability. Formally, it writes as follows:

$$\max_{\pi \in \Pi_b} \rho(\pi, \widehat{M}), \text{ where } \Pi_b \subset \{\pi \in \Pi \text{ s.t. } \forall M \in \Xi, \rho(\pi, M) \geq \rho(\pi_b, M) - \zeta\}. \quad (1)$$

Clearly, the larger the Π_b the greater the potential policy improvement and the harder the optimization problem. By appropriately balancing those opposite effects, SPIBB makes searching for an efficient and provably-safe policy tractable in terms of computer time, while allowing for potentially substantial policy improvements. To construct Π_b, we let $N_{\mathcal{D}}(x,a)$ denote the count of state-action pair (x,a) in $\mathcal{D}$. We then define a *bootstrapped set* $\mathfrak{B} \subset \mathcal{X} \times \mathcal{A}$ containing all the pairs (x,a) whose counts are smaller than a fixed parameter $N_\wedge$. In other words, $\mathfrak{B}$ contains the state-action pairs (x,a) for which the uncertainty is high. Π_b then denotes the set of policies π that verify:

$$\forall (x,a) \in \mathfrak{B}, \pi(a|x) = \pi_b(a|x). \tag{2}$$

In the uncertain pairs, for which relying on the observed data could potentially be risky, SPIBB leans on the baseline by copying its probability to take action a. In the others, SPIBB follows classic policy optimization techniques. For finite MDPs, this may be achieved in a model-based manner by explicitly computing the MDP model $\widehat{M}$, constructing the set of allowed policies Π_b and finally searching for the Π_b-optimal policy $\pi^{\odot}_{spibb}$ in $\widehat{M}$ using policy iteration over Π_b [9, 15]. [12] prove that Π_b-SPIBB converges to a Π_b-optimal policy $\pi^{\odot}_{spibb}$ in $\widehat{M}$, and that $\pi^{\odot}_{spibb}$ is a safe policy improvement over the baseline in the true MDP M^*. They also apply SPIBB to a gridworld example and show its benefits over existing algorithms. In the following, we describe an extension of their method to function approximators.

SPIBB-DQN: DQN [13] successfully applies Q-learning to complex video games that require deep neural networks. The method uses a variety of techniques but fundamentally consists in iteratively applying the Bellman operator to learn the Q-values of the environment. DQN can easily be extended to a batch setting by replacing the experience memory used in the original algorithm with the batch we are training on. Similarly, the SPIBB policy optimization described above may be achieved in a model-free manner by sampling transitions $\langle x_j, a_j, r_j, x'_j \rangle$ from the dataset and fitting the Q-function $Q^{(t+1)}(x_j, a_j)$ to the following target $y_j^{(t)}$:

$$\begin{aligned} y_j^{(t)} &= r_j + \gamma \max_{\pi \in \Pi_b} \sum_{a' \in \mathcal{A}} \pi(a'|x'_j) Q^{(t)}(x'_j, a') \\ &= r_j + \gamma \sum_{a' | (x'_j, a') \in \mathfrak{B}} \pi_b(a'|x'_j) Q^{(t)}(x'_j, a') + \gamma \left(\sum_{a' | (x'_j, a') \notin \mathfrak{B}} \pi_b(a'|x'_j) \right) \max_{(x'_j, a') \notin \mathfrak{B}} Q^{(t)}(x'_j, a'). \end{aligned}$$

The first term r_j is the immediate reward observed during the recorded transition, the second term is the return estimate of the bootstrapped actions (where the trained policy is constrained to the baseline policy), and the third term is the return estimate maximized over the non-bootstrapped actions. We call this algorithm SPIBB-DQN as it is equivalent to DQN fitted to $y_j^{(t)}$. Note that the computation of the SPIBB targets requires the computation of the bootstrapped set $\mathfrak{B}$, which relies on an estimate of the state-action counts $\widetilde{N}_{\mathcal{D}}(x,a)$, sometimes called pseudo-counts [1, 7, 3]. The safety of SPIBB-DQN increases with $N_\wedge$. For $N_\wedge = 0$, we observe that $\mathfrak{B} = \emptyset$, and SPIBB-DQN amounts to the original DQN. As $N_\wedge \to \infty$, we see that $\mathfrak{B} = \mathcal{X} \times \mathcal{A}$, thus SPIBB-DQN becomes fully conservative and simply reproduces π_b. Intermediate values of $N_\wedge$ allow a balance between these two extreme cases.

Among existing batch RL algorithms, and to the best of our knowledge, only RaMDP [14] can straightforwardly be extended to the function approximation setting. RaMDP stands for Reward-adjusted MDP and consists in modifying the observed reward by $r_j \leftarrow r_j - \kappa / \sqrt{\widetilde{N}_{\mathcal{D}}(x_j, a_j)}$ (κ is a hyperparameter of their model).

3 SPIBB-DQN empirical evaluation

The performance of Batch RL algorithms can vary greatly from one dataset to another. To properly evaluate SPIBB-DQN and its ability to produce policies that reliably outperform the baseline, we randomly generate 20 fixed-size datasets and train 15 policies on each dataset for each algorithm and for various hyper-parameter values. The algorithms are then evaluated using the mean performance and conditional value at risk performance (CVaR, also called expected shortfall) of the policies they produce. The $X\%$-CVaR is the mean performance over the $X\%$ worst runs, it is commonly used to assess the robustness of algorithms.

For the sake of simplicity and to be able to repeat several runs of each experiment efficiently, instead of applying costly and often finicky pseudo-count methods from the literature [1, 7, 3], we consider here a pseudo-count heuristic based on the Euclidean distance between states, and tasks where it makes sense to do so. The pseudo-count of a state-action (x,a) is defined as the sum of its similarity with the state-action pairs (x_i, a_i) found in the dataset. The similarity between (x,a) and (x_i,a_i) is equal to 0 if $a_i \neq a$, and to $\max(0, 1 - 5d(x, x_i))$ otherwise (where $d(\cdot,\cdot)$ is the Euclidean distance between two states).

We first consider a helicopter navigation task (see Figure 1(a)). The helicopter starts from a random position in the teal area, with a random initial velocity. The 9 available actions consist in applying thrust: backward, no, or forward acceleration, along the two dimensions. The episode terminates when the velocity exceeds some maximal value, in which case it gets a -1 reward, or when the helicopter leaves the blue area, in which case it gets a reward as chromatically indicated on Figure 1(a). The dynamics of the domain follow the basic laws of physics with a Gaussian centered additive noise both on the position and the velocity. To train our algorithms, we use a discount factor $\gamma = 0.9$, but report the undiscounted final reward. The baseline is generated as follows: we first train a policy with online DQN, stop before full convergence and then apply a softmax on the obtained Q-network. Our experiments consist in 300 runs on SPIBB-DQN with a range of $N_\wedge$ values and for different dataset sizes. SPIBB-DQN with $N_\wedge = 0$ is equivalent to vanilla DQN. We also perform a hyper-parameter search for RaMDP on 10k-transition datasets, with κ values ranging from 0.001 to 1000. To reduce the computational load, we only performed 75 runs per value. We also display $\kappa = 0$, which amounts to vanilla DQN. Figure 2(b) shows that, although it slightly improves over DQN, RaMDP is rather limited and stays far below the baseline.

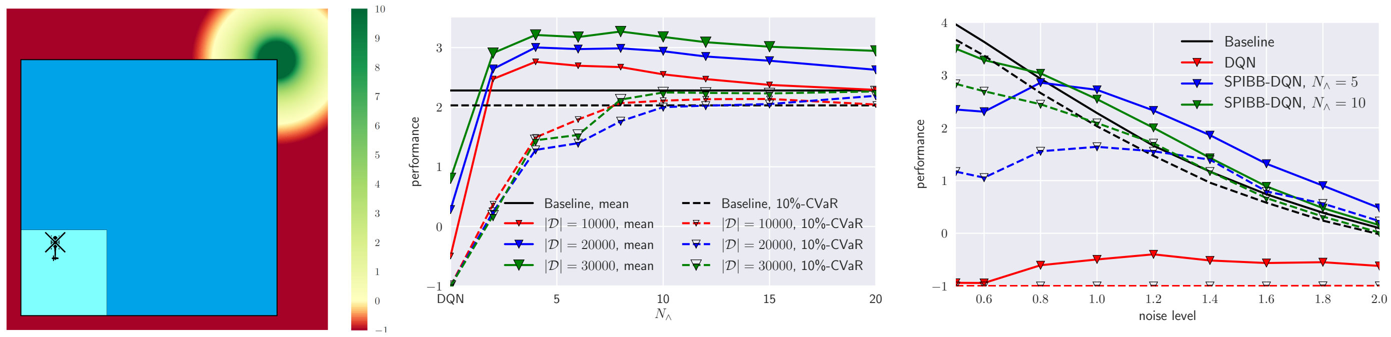
(a) Helicopter environment. (b) Mean and 10%-CVaR vs $N_\wedge$. (c) Performance vs noise factor.

Figure 1: (a) Illustration of the environment. (b) Mean and 10%-CVaR performance as a function of $N_\wedge$ for three dataset sizes. (c) Mean and 10%-CVaR performance for the baseline, vanilla DQN, SPIBB-DQN with $N_\wedge = 5, 10$, as a function of the transition noise factor.

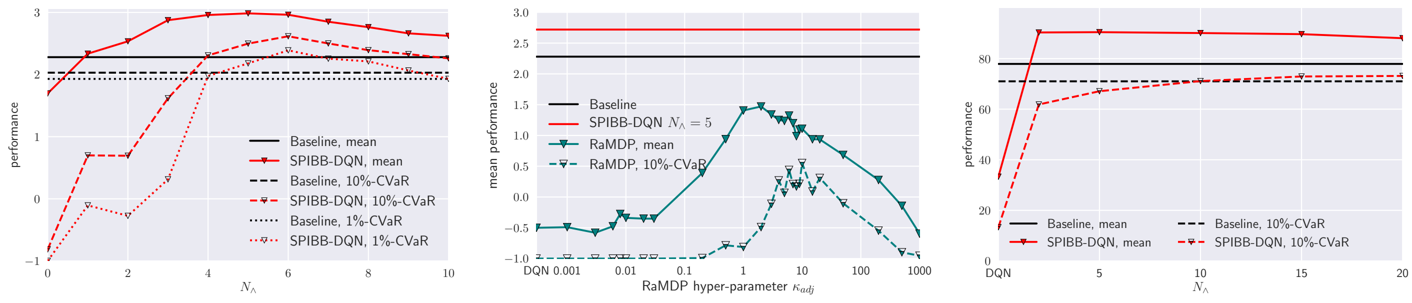
(a) Mean, 10%-CVaR and 1%-CVaR vs $N_\wedge$. (b) RaMDP Performance vs κ. (c) Performance on CartPole.

Figure 2: (a) Mean, 10%-CVaR and 1%-CVaR as a function of $N_\wedge$ for a single 10k dataset. (b) Mean and 10%-CVaR performance of RaMDP for multiple values of κ over the 20 10k-datasets from Figure 1. (c) Mean and 10%-CVaR performance on CartPole.

Figure 1(b) displays the mean and 10%-CVaR performances in function of $N_\wedge$ for three dataset sizes (10k, 20k, and 30k). We observe that vanilla DQN ($N_\wedge = 0$) significantly worsens the baseline in mean and achieves the worst possible 10%-CVaR performance. SPIBB-DQN not only significantly improves the baseline in mean performance for $N_\wedge \geq 1$, but also in 10%-CVaR when $N_\wedge \geq 8$. The discerning reader might wonder about the CVaR curve for the baseline. It is explained by the fact that the evaluation of the policies are not exact. The curve accounts for the evaluation errors, errors also obviously encountered with the trained policies. We performed an additional experiment. Keeping the baseline identical, we trained on 10k-transitions datasets obtained from environments with different transition noises. Figure 1(c) shows the mean and 10%-CVaR performances for the baseline, vanilla DQN, and SPIBB-DQN with $N_\wedge \in \{5, 10\}$. First, we observe that vanilla DQN performs abysmally. Second, we see that the baseline quickly gets more efficient when the noise is removed making the safe policy improvement task harder for SPIBB-DQN. As SPIBB is efficient at dealing with stochasticity, the noise attenuation reduces its usefulness. Third, as we get to higher noise factors, the stochasticity becomes too high to efficiently aim at the goal, but SPIBB-DQN still succeeds at safely improving the baseline.

Before starting the experiments reported above, we led preliminary experiments with a single 10k-transitions dataset. We found out, and report on Figure 2(a), that vanilla DQN produces very different Q-networks (and therefore very different policies) for different random seeds, even on the same dataset. It is worth noticing a posteriori that this dataset was actually favorable to DQN (mean performance of 1.7 on this dataset vs. -0.5 averaged over 20 datasets, Figure 1(b)), but that DQN's reliability is still very low. In contrast, SPIBB-DQN shows stability for $N_\wedge \geq 4$. We also report the 1%-CVaR performance (quite heavy computationally to accurately estimate) and see that there too, SPIBB-DQN performs well.

We also apply SPIBB-DQN to the CartPole environment from OpenAI gym [2]. Similarly to above, we first train a policy on the environment and build a baseline from it by applying a softmax to the learnt Q-values. To make the task more challenging once the baseline has been obtained, we add some stochasticity to the environment by acting randomly 50% of the time. With that added noise, the baseline gets an average score of approximately 78. SPIBB-DQN reaches values over 90 while remaining safe. Standard DQN gets a score of 33 and a 10%-CVaR of 13. Results can be found on Figure 2(c), for datasets of size 10k. We also applied our algorithm to Pendulum with marginal improvements (not shown here).

4 Conclusion and future work

In this paper, we study the problem of safe Batch Reinforcement Learning with function approximation. We develop a model-free version of SPIBB and demonstrate its performance on two domains. SPIBB-DQN is the first deep batch algorithm allowing policy improvement in a safe manner. Future work involves extending our results to more complex domains, *e.g.* with pixel state spaces such as Atari games. This would, in particular, imply adapting existing pseudo-counts techniques to the batch setting.

References

[1] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2016.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[4] Erick Delage and Shie Mannor. Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 2010.

[5] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1329–1338, 2016.

[6] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[7] Lior Fox, Leshem Choshen, and Yonatan Loewenstein. Dora the explorer: Directed outreaching reinforcement action-selection. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[8] Aude Genevay and Romain Laroche. Transfer learning for user adaptation in spoken dialogue systems. In *Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2016.

[9] Ronald A Howard. Dynamic programming. *Management Science*, 1966.

[10] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*. Springer, 2012.

[11] Romain Laroche and Raphaël Féraud. Reinforcement learning algorithm selection. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[12] Romain Laroche and Paul Trichelair. Safe policy improvement with baseline bootstrapping. In *Proceedings of the 14th European Workshop on Reinforcement Learning*, 2018.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[14] Marek Petrik, Mohammad Ghavamzadeh, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Proceedings of the 29th Advances in Neural Information Processing Systems (NIPS)*, 2016.

[15] Martin L Puterman and Shelby L Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 1979.

[16] Thiago D. Simao and Matthijs T. J. Spaan. Safe policy improvement with baseline bootstrapping in factored environments. In *Proceedings of the 33th AAAI Conference on Artificial Intelligence*, 2019.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[18] István Szita and András Lőrincz. The many faces of optimism: a unifying approach. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

[19] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[20] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.