Parameterized projected Bellman operator

Théo Vincent [1 2] **Alberto Maria Metelli** [3] **Jan Peters** [1 2 4] **Marcello Restelli** [3] **Carlo D'Eramo** [4 5]

Abstract

The Bellman operator is a cornerstone of reinforcement learning (RL), widely used from traditional value-based methods to modern actor-critic approaches. In problems with unknown models, the Bellman operator is estimated via transition samples that strongly determine its behavior, as uninformative samples can result in negligible updates or long detours before reaching the fixed point. In this paper, we introduce the novel idea of an operator that acts on the parameters of action-value function approximators. Our novel operator can obtain a sequence of action-value function parameters that progressively approaches the ones of the optimal action-value function. This means that we merge the traditional two-step procedure consisting of applying the Bellman operator and subsequently projecting onto the space of action-value function. For this reason, we call our novel operator *projected* Bellman operator (PBO). We formulate an optimization problem to learn PBOs for generic sequential decision-making problems, and we analyze the PBO properties in two representative classes of RL problems. Furthermore, we study the use of PBO under the lens of the approximate value iteration framework, devising algorithmic implementations to learn PBOs in both offline and online settings, resorting to neural network regression. Eventually, we empirically evince how PBO can overcome the limitations of classical methods, opening up new research directions as a novel paradigm in RL.

[1]Department of Computer Science, TU Darmstadt, Germany [2]German Research Center for AI (DFKI), Research Department: Systems AI for Robot Learning, Germany [3]Department of Electronic, Computer Science and Bioengineering, Politecnico di Milano, Italy [4]Hessian.ai, Germany [5]Center for Artificial Intelligence and Data Science, University of Würzburg, Germany. Correspondence to: Théo Vincent <vincent@robot-learning.de>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

1. Introduction

Value-based reinforcement learning (RL) is a popular class of algorithms for solving sequential decision-making problems with unknown dynamics (Sutton & Barto, 2018). For a given problem, value-based algorithms aim at obtaining the most accurate estimate of the expected return from each state, i.e., a value function. For instance, the well-known value-iteration algorithm computes value functions by iterated applications of the Bellman operator (Bellman, 1966), of which the true value function is the fixed point. Although the Bellman operator can be applied in an exact way in dynamic programming, it is typically estimated from samples *at each application* when dealing with problems with unknown models, i.e., empirical Bellman operator (Watkins, 1989; Bertsekas, 2019). Intuitively, the dependence of the empirical version of value iteration on the samples has an impact on the efficiency of the algorithms and on the quality of the obtained estimated value function, which becomes accentuated when solving continuous problems that require function approximation, e.g., approximate value iteration (AVI) (Munos, 2005; Munos & Szepesvári, 2008). Moreover, in AVI approaches, costly function approximation steps are needed to project the output of the Bellman operator back to the considered action-value functional space.

In this paper, we introduce the novel notion of *projected Bellman operator* (PBO), which is a function $\Lambda : \Omega \to \Omega$ defined on parameters $\boldsymbol{\omega} \in \Omega$ of the action-value function approximator. Contrarily to the standard (empirical) Bellman operator, which uses action-value functions Q_k to compute targets that are then projected to obtain Q_{k+1}, our PBO uses the parameters of the action-value function to directly compute updated parameters $\boldsymbol{\omega}_{k+1} = \Lambda(\boldsymbol{\omega}_k)$ (Figure 1). We additionally provide a perspective on AVI where, instead of exploiting samples to iterate over the application of the empirical Bellman operator and its subsequent projection step, we leverage them to learn an approximation of PBO, which we call *parameterized* PBO. The crucial ad-

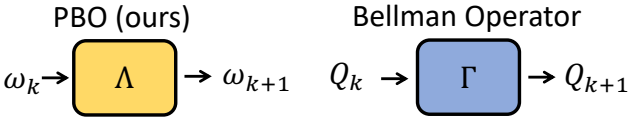
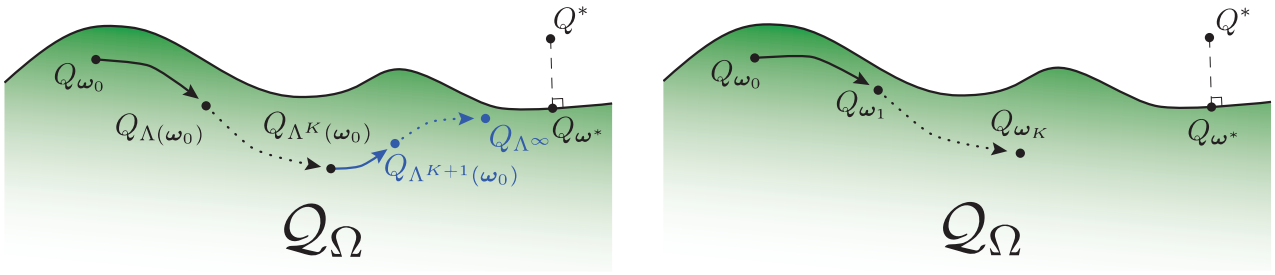


Figure 1: PBO operates on value function parameters.



(a) PBO (ours) (b) AVI

Figure 2: Behavior of our PBO and approximate value iteration (AVI) in the space of value functions. Q^*, $Q_{\boldsymbol{\omega}^*}$, and Q_{Λ^∞}, are respectively the optimal value function, its projection on the parametric space, and the fixed point of PBO.

vantages of our approach are twofold: (i) the output of PBO always belongs to the considered action-value functional space, avoiding, therefore, the costly projection step needed when using the Bellman operator coupled with function approximation, and (ii) PBO is applicable for an arbitrary number of iterations without using further samples, as visualized in Figure 2. Starting from initial parameters $\boldsymbol{\omega}_0$, AVI approaches obtain consecutive approximations of the action-value function $Q_{\boldsymbol{\omega}_k}$ by applying the Bellman operator iteratively over samples (Figure 2b). Instead, we make use of the samples to learn the PBO only. Then, starting from initial parameters $\boldsymbol{\omega}_0$, PBO can produce a chain of action-value function parameters of arbitrary length (blue lines in Figure 2a), without requiring further samples. This means that an accurate approximation of PBO can compute optimal action-value function parameters starting from any initial parameters in the chosen space.

Contributions. In the following, we formally introduce PBO, and formulate an optimization problem and algorithmic implementations to obtain it in offline and online RL problems. Thus, our contribution is threefold: (i) we introduce the notion of projected Bellman operator (PBO); (ii) we derive an optimization problem to approximate PBOs in generic Markov decision processes (MDPs); (iii) we introduce two novel algorithms for offline and online RL to solve this optimization problem, showing their advantages over related baselines on heterogeneous RL problems.

2. Related works

Several works in the literature proposed variants of the standard Bellman operator to induce some desired behavior. We revise these approaches, noting that all of them act on the space of action-value functions, thus needing a costly projection step onto the considered functional space. Conversely, our work is, to the best of our knowledge, the first attempt to obtain an alternative Bellman operator that directly acts on the parameters of action-value functions.

Bellman operator variations. Variants of the Bellman operator are widely studied for entropy-regularized MDPs (Neu et al., 2017; Geist et al., 2019; Belousov & Peters, 2019). The softmax (Haarnoja et al., 2017; Song et al., 2019), mellowmax (Asadi & Littman, 2017), and optimistic (Tosatto et al., 2019) operators are all examples of variants of the Bellman operator to obtain maximum-entropy exploratory policies. Besides favoring exploration, other approaches address the limitations of the standard Bellman operator. For instance, the consistent Bellman operator (Bellemare et al., 2016) is a modified operator that addresses the problem of inconsistency of the optimal action-value functions for suboptimal actions. The distributional Bellman operator (Bellemare et al., 2017) enables operating on the whole return distribution, instead of its expectation, i.e., the value function (Bellemare et al., 2023). Furthermore, the logistic Bellman operator uses a logistic loss to solve a convex linear programming problem to find optimal value functions (Bas-Serrano et al., 2021). Finally, the Bayesian Bellman operator is employed to infer a posterior over Bellman operators centered on the true one (Fellows et al., 2021). Note that our PBO can be seamlessly applied to any of these variants of the standard Bellman operator. Finally, we recognize that learning an approximation of the Bellman operator shares some similarities with learning a reward-transition model in reinforcement learning. However, we point out that our approach is profoundly different, as we map action-value parameters to other action-value parameters, conversely to model-based reinforcement learning which maps states and actions to rewards and next states.

Operator learning. Literature in operator learning is mostly focused on supervised learning, with methods for learning operators over vector spaces (Micchelli & Pontil, 2005) and parametric approaches for learning non-linear operators (Chen & Chen, 1995), with a resurgence of recent contributions in deep learning. For example, Kovachki et al. (2021) learn mappings between infinite function spaces with deep neural networks, or Kissas et al. (2022) apply an attention mechanism to learn correlations in the target function for efficient operator learning. We note that our work on the

learning of the Bellman operator in reinforcement learning is orthogonal to methods for operator learning in supervised learning, and could potentially benefit from advanced techniques in the literature.

3. Preliminaries

We consider discounted Markov decision processes (MDPs) defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a measurable state space, $\mathcal{A}$ is a finite measurable action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[1] is the transition kernel of the dynamics of the system, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1)$ is a discount factor (Puterman, 1990). A deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function mapping a state to an action, inducing a value function $V^\pi(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t \mathcal{R}(S_t, \pi(S_t)) | S_0 = s\right]$ representing the expected cumulative discounted reward starting in state s and following policy π thereafter. Similarly, the action-value function $Q^\pi(s, a) \triangleq \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t \mathcal{R}(S_t, A_t) | S_0 = s, A_0 = a, A_t = \pi(S_t)\right]$ is the expected discounted cumulative reward executing action a in state s, following policy π thereafter. RL aims to find an optimal policy π^* yielding the optimal value function $V^*(s) \triangleq \max_{\pi : \mathcal{S} \to \mathcal{A}} V^\pi(s)$ for every $s \in \mathcal{S}$ (Puterman, 1990). The (optimal) Bellman operator Γ^* is a fundamental tool in RL for obtaining optimal policies, and it is defined as:

$$(\Gamma^* Q)(s, a) \triangleq \mathcal{R}(s, a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s' | s, a) \max_{a' \in \mathcal{A}} Q(s', a'), \quad (1)$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. It is well-known that Bellman operators are contraction mappings in L_∞-norm, such that their iterative application leads to the fixed point $\Gamma^* Q^* = Q^*$ in the limit (Bertsekas, 2015). We consider the use of function approximation to represent value functions and denote Ω the space of their parameters. Thus, we define $\mathcal{Q}_\Omega = \{Q_{\boldsymbol{\omega}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} | \boldsymbol{\omega} \in \Omega\}$ as the set of value functions representable by parameters of Ω.

4. Projected Bellman operator

The application of the Bellman operator in RL requires transition samples (Equation 1) (Munos, 2003; 2005; Munos & Szepesvári, 2008). The impact of this requirement is twofold: (i) the behavior of the Bellman operator strictly depends on the quality of the samples, thus, a poor sample selection can result in negligible updates and long detours before convergence; (ii) the direction of the update is determined exclusively by the current samples, meaning that the empirical Bellman operator has no memory of previously observed samples (Bellemare et al., 2016; Agarwal et al., 2019; Fellows et al., 2021). Ideally, we want to obtain an operator that mirrors the behavior of the true Bellman operator while being independent of the transition samples and extrapolating from previous experience. Thus, we introduce the novel *projected Bellman operator* (PBO), which we define as follows.

Definition 4.1. Let $\mathcal{Q}_\Omega = \{Q_{\boldsymbol{\omega}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} | \boldsymbol{\omega} \in \Omega\}$ be a function approximation space for the action-value function, induced by the parameter space Ω. A projected Bellman operator (PBO) is a function $\Lambda : \Omega \to \Omega$, such that

$$\Lambda \in \arg\min_{\Lambda : \Omega \to \Omega} \mathbb{E}_{(s,a) \sim \rho, \boldsymbol{\omega} \sim \nu} \left(\Gamma^* Q_{\boldsymbol{\omega}}(s, a) - Q_{\Lambda(\boldsymbol{\omega})}(s, a)\right)^2, \quad (2)$$

for state-action and parameter distributions ρ and ν.

Note that Γ^* is the regular optimal Bellman operator on $\mathcal{Q}_\Omega$. Conversely, PBO is an operator Λ acting on action-value function parameters $\boldsymbol{\omega} \in \Omega$. In other words, this definition states that a PBO is the mapping $\Omega \to \Omega$ that most closely emulates the behavior of the regular optimal Bellman operator Γ^*.

4.1. Learning projected Bellman operators

The PBO is unknown and has to be estimated from samples. We propose to approximate the unknown PBO with a *parameterized PBO* $\Lambda_{\boldsymbol{\phi}}$ differentiable w.r.t. its parameters $\boldsymbol{\phi} \in \Phi$, enabling the use of gradient descent on a loss function[2]. A first straightforward idea would be to learn an operator that minimizes an empirical version of the loss in the definition of PBO (Equation 2), for given datasets of parameters $\boldsymbol{\omega} \in \mathcal{W}$ and transitions $(s, a, r, s') \in \mathcal{D}$. However, this approach would disregard the crucial opportunity of performing multiple applications of the operator. This means that we can apply PBO on the parameters $\boldsymbol{\omega} \in \mathcal{W}$ for an arbitrary number of iterations, thus augmenting the dataset of parameters with sequences of parameters generated by PBO. We leverage this key insight to formulate the loss

$$\mathcal{L}_{\Lambda_{\boldsymbol{\phi}}} = \sum_{k=1}^{K} \sum_{\substack{(s,a) \in \mathcal{D} \\ \boldsymbol{\omega} \in \mathcal{W}}} \left(\Gamma^* Q_{\Lambda_{\boldsymbol{\phi}}^{k-1}(\boldsymbol{\omega})}(s, a) - Q_{\Lambda_{\boldsymbol{\phi}}^{k}(\boldsymbol{\omega})}(s, a)\right)^2, \quad (3)$$

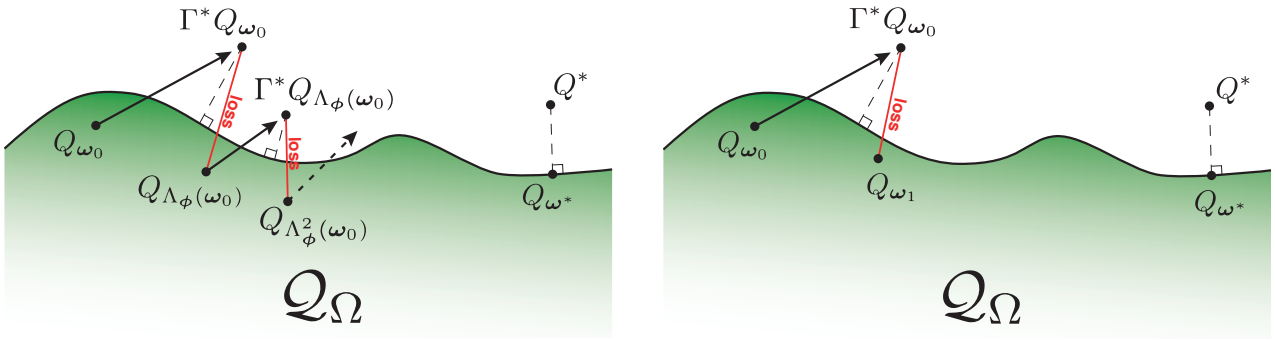
where K is an arbitrary number of optimal Bellman operator iterations. An additional idea is to add a term that corresponds to an infinite number of iterations, i.e., the fixed point, when possible

$$\mathcal{L}_{\Lambda_{\boldsymbol{\phi}}^\infty} = \mathcal{L}_{\Lambda_{\boldsymbol{\phi}}} + \sum_{(s,a) \in \mathcal{D}} \underbrace{\left(\Gamma^* Q_{\Lambda_{\boldsymbol{\phi}}^\infty}(s, a) - Q_{\Lambda_{\boldsymbol{\phi}}^\infty}(s, a)\right)^2}_{\text{Fixed point term}}, \quad (4)$$

where $\Lambda_{\boldsymbol{\phi}}^\infty$ is the fixed point of the parameterized PBO. Note that the addition of the fixed point term is only possible for

[1] $\Delta(\mathcal{X})$ denotes the set of probability measures over the set $\mathcal{X}$.

[2] For ease of presentation, we use PBO and parameterized PBO interchangeably whenever clear from the context.



(a) ProFQI (ours) (b) FQI

Figure 3: Behavior of ProFQI and FQI in the function space $\mathcal{Q}_\Omega$ for *one* iteration. The dashed lines are projections in $\mathcal{Q}_\Omega$.

classes of parameterized PBOs where the fixed point can be computed and the result is differentiable in the parameters ϕ, as discussed in Section 5.

4.2. An approximate value iteration perspective

We devise two approximate value iteration (AVI) (Farahmand, 2011) algorithms to learn PBOs in offline and online RL, by minimizing the loss (3) or (4). Our algorithms can be seen as variants of fitted Q-iteration (FQI) (Ernst et al., 2005) and of deep Q-network (DQN) (Mnih et al., 2015); thus, we call them *projected fitted Q-iteration* (ProFQI) and *projected deep Q-network* (ProDQN). Figure 3 illustrates the differences between our approaches and the baseline, by taking ProFQI and FQI as examples and comparing how they behave in the space of all possible action-value functions $\mathcal{Q}_\Omega$. The main difference is that ProFQI generates a sequence of action-value parameters, while FQI generates a sequence of action-value functions that need to be projected onto the functional space. Moreover, ProFQI incorporates the sequence of parameters, generated by multiple applications of PBO, in the loss function (see also Equation 3). This results in a richer loss than FQI, which can only consider one application of the Bellman operator at a time, resulting in better approximation (see red lines in Figure 3a and 3b). This ability of PBO to iterate for an arbitrary number of times also enables us to theoretically prove the advantages of ProFQI by leveraging established results in AVI.

Theorem 4.2. *(See Theorem 3.4 of Farahmand (2011)) Let $K \in \mathbb{N}^*$, ρ, ν two distribution probabilities over $\mathcal{S} \times \mathcal{A}$. For any sequence $(Q_k)_{k=0}^K \subset B(\mathcal{S} \times \mathcal{A}, R_\gamma)$ where R_γ depends on reward function and discount factor, we have*

$$||Q^* - Q^{\pi_K}||_{1,\rho} \leq C_{K,\gamma,R_\gamma} \tag{5}$$
$$+ \inf_{r \in [0,1]} F(r; K, \rho, \gamma) \left(\sum_{k=1}^{K} \alpha_k^{2r} \overbrace{\underbrace{||\Gamma^* Q_{k-1} - Q_k||_{2,\nu}^2}_{FQI}}^{ProFQI} \right)^{\frac{1}{2}},$$

Algorithm 1 Projected FQI & Projected DQN

Inputs:
- samples $\mathcal{D} = \{\langle s_j, a_j, r_j, s'_j \rangle\}_{j=1}^J$;
- parameters $\mathcal{W} = \{\omega_l\}_{l=1}^L$;
- #Bellman iterations K;
- initial parameters ϕ of parameterized PBO Λ_ϕ;
- #Epochs E.

for $e \in \{1, \dots, E\}$ **do**
  $\bar{\phi} = \phi$
  for some training steps **do**
    Collect samples $\mathcal{D}'$ with policy given by $Q_{\Lambda^K_{\bar{\phi}}(\omega)}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'$
    Gradient descent over parameters ϕ minimizing loss (3) or (4) using $\mathcal{D}$ or a batch of $\mathcal{D}$ and $\mathcal{W}$.
  end for
end for
Return: Parameters ϕ of parameterized PBO Λ_ϕ

where α_k and C_{K,γ,R_γ} do not depend on the sequence $(Q_k)_{k=0}^K$. $F(r; K, \rho, \gamma)$ relies on the concentrability coefficients of the greedy policies w.r.t. the value functions.

This theorem shows that the distance between the value function and the optimal value function depends on the approximation errors for all the iterations. The loss of FQI contains only one term of the sum, while the loss of ProFQI contains the entire sum. In ProFQI, the terms are summed up with equal weights, which is not the case for Equation (5). Additionally, Algorithm 1 describes the steps of our algorithms for learning PBO, highlighting the additional steps required for the online setting (i.e., ProDQN). Both ProFQI and ProDQN are given initial randomly sampled datasets of transitions and parameters of PBO. As an online algorithm, ProDQN periodically adds new transitions to the dataset by executing a policy derived from the action-value function obtained by applying the current approximation of PBO for K times. For stability reasons, we perform gradient

descent steps only on the parameters ϕ of $Q_{\Lambda_\phi^k}$ in the losses (Equations 3 and 4), excluding the ones corresponding to the target $\Gamma^* Q_{\Lambda_\phi^{k-1}}$ of the loss, as commonly done in semi-gradient methods (Sutton & Barto, 2018). Similar to Mnih et al. (2015), the target parameters are updated periodically after an arbitrary number of iterations. Soft updates, e.g., Polyak averaging (Lillicrap et al., 2015), can also be used.

5. Analysis of projected Bellman operator

We analyze the properties of PBO for two representative classes of RL problems, namely (i) finite MDPs and (ii) linear quadratic regulation (Bradtke, 1992; Pang & Jiang, 2021)[3]. Proofs of the following results are in Appendix B.

5.1. Finite Markov decision processes

Let us consider finite state and action spaces of cardinality N and M, respectively, and a tabular setting with $\Omega = \mathbb{R}^{N \cdot M}$. Given the use of tabular approximation, it is intuitive that each entry of the table can be modeled with a different single parameter, i.e., there is a bijection between $\mathcal{Q}_\Omega$ and Ω, which allows us to write, for ease of notation, the parameters of the action-value function as Q instead of $\boldsymbol{\omega}$.

Proposition 5.1. *The PBO exists and it is equal to the optimal Bellman operator*

$$\Lambda^*(Q) = R + \gamma P \max_{a \in \mathcal{A}} Q(\cdot, a). \tag{6}$$

Note that PBO for finite MDPs is a γ-contraction mapping for the L_∞-norm, like an optimal Bellman operator. As an example of finite MDP, we consider the chain-walk environment in Figure 5, with a chain of length $N = 20$. We parameterize value functions as tables to leverage our theoretical results on finite MDPs. In Figure 4, we show the ℓ_2-norm of the difference between the optimal action-value function and the action-value function computed with FQI and ProFQI. We consider three different values of Bellman iterations, namely $K = \{2, 5, 15\}$. For FQI, the K iterations are the regular iterations where the empirical Bellman operator is applied once on the current approximation of the action-value function. For ProFQI, K is the number of iterations included in the PBO loss. This ensures a fair comparison given that both methods have access to the same number of Bellman iterations. Once PBO is trained, we apply it for different numbers of iterations $k \geq K$, on given action-value function parameters. In Figure 4, ProFQI uses a linear approximation of PBO trained with the loss (3), ProFQI$_\infty$ uses a linear approximation of PBO trained with the loss (4), ProFQI$_{\text{chain}}$ uses the closed-form PBO in Equation (6) considering R and P as unknown parameters to learn, and PBO indicates the use of the same closed-form solution assuming known R and P. First, note that in this basic setting FQI and PBO minimize the same objective function. Obviously, the difference is that we can always apply the closed-form solution for more iterations than FQI, achieving a better approximation of the optimal action-value function, as shown in Figure 4. For the three variants of ProFQI, we observe that the approximation error decreases as the number of PBO iterations increases, evincing that PBO accurately emulates the true Bellman operator. In the case of $K = 2$ and $K = 5$, we see that ProFQI$_\infty$ and ProFQI$_{\text{chain}}$ obtain a better approximation of the action-value function compared to ProFQI, thanks to, respectively, the inclusion of fixed point in the loss and the use of the closed-form solution. Interestingly, in the case of $K = 15$, ProFQI$_\infty$ obtains a slightly worse approximation than ProFQI. We impute this behavior to the fact that when the linear approximation is inadequate for modeling PBO, adding the fixed point in the loss could harm the estimate.

5.2. Linear quadratic regulation

Now, we consider the continuous MDPs class of linear quadratic regulator (LQR) with $\mathcal{S} = \mathcal{A} = \mathbb{R}$. The transition model $\mathcal{P}(s, a) = As + Ba$ is deterministic and the reward function $\mathcal{R}(s, a) = Qs^2 + 2Ssa + Ra^2$ is quadratic, where A, B, Q, S and R, are constants inherent to the MDP. We choose to parameterize the action-value functions with a 2-dimensional parameter vector $\mathcal{Q}_\Omega = \{(s, a) \mapsto Gs^2 + 2Isa + Ma^2 | (G, I) \in \mathbb{R}^2\}$ where M is a chosen constant, for visualization purposes.

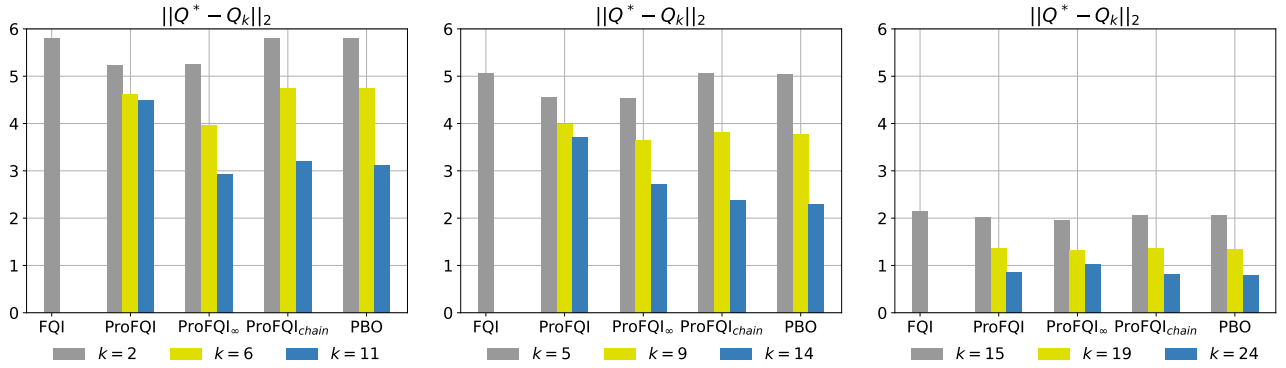
Proposition 5.2. *PBO exists and for any $\boldsymbol{\omega} \in \mathbb{R}^2$ its closed form[4] is given by:*

$$\Lambda^* : \boldsymbol{\omega} = \begin{bmatrix} G \\ I \end{bmatrix} \mapsto \begin{bmatrix} Q + A^2(G - \frac{I^2}{M}) \\ S + AB(G - \frac{I^2}{M}) \end{bmatrix}. \tag{7}$$

We leverage our theoretical analysis for LQR (Bradtke, 1992; Pang & Jiang, 2021) by parameterizing value functions accordingly, and we conduct a similar analysis to the one for chain-walk. This time, we evaluate the distance between the optimal action-value function parameters, which can be computed analytically, and the estimated ones. We use the closed-form solution obtained in Equation 7 assuming the parameters (A, B, Q, S) known (PBO), and unknown (ProFQI$_{\text{LQR}}$). Figure 6 confirms the pattern observed on chain-walk. ProFQI and ProFQI$_\infty$ obtain a better approximation than FQI, which is reasonably worse than ProFQI$_{\text{LQR}}$ and PBO. We also observe that ProFQI$_{\text{LQR}}$ obtains a significantly better approximation than the other variants for a large number of iterations (blue bars), confirming the advantages of exploiting the closed-form solu-

[3] An additional analysis of PBO in low-rank MDPs (Agarwal et al., 2020; Sekhari et al., 2021) can be found in Appendix A.

[4] Under an assumption over the sample distribution, see in Appendix B



(a) $K = 2$. (b) $K = 5$. (c) $K = 15$.

Figure 4: ℓ_2-norm of the difference between the optimal action-value function and the approximated action-value function on chain-walk. Results are averaged over 20 seeds.

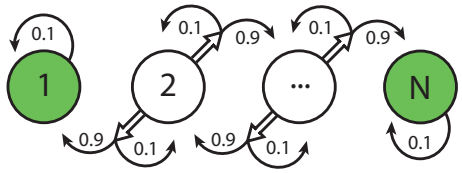


Figure 5: The chain-walk from Munos (2003). Reward is always 0, but 1 in green states.

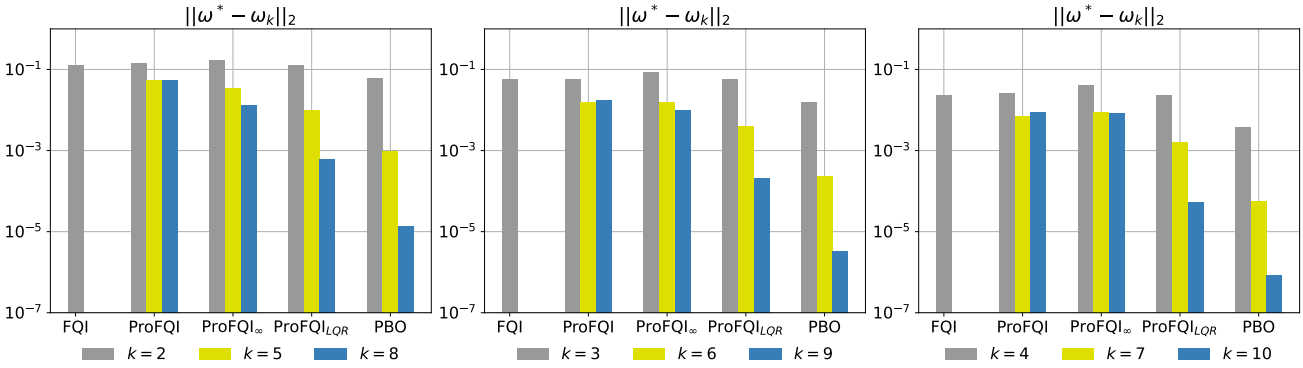
tion of PBO for finite MDPs. We additionally show the sequence of parameters corresponding to the iterations of FQI, ProFQI$_{\text{LQR}}$, and ProFQI, in Figure 7. The training is done with $K = 2$. The sequence starts from the chosen initial parameters $(G, I) = (0, 0)$ for all algorithms, and proceeds towards the optimal parameters, which are computed analytically. Both ProFQI and ProFQI$_{\text{LQR}}$ apply the PBO learned after the $K = 2$ iterations, for 8 iterations; thus, the sequence of parameters for both algorithms is composed of 8 points each, while FQI has 2. It is clear that the parameters found by FQI are considerably further to the target parameters, than the ones found by both ProFQI and ProFQI$_{\text{LQR}}$. In particular, the latter gets the closest to the target, in line with the results in Figure 6, again evincing the accuracy of the learned PBO and the benefit of performing multiple applications of it.

6. Experiments

We challenge our PBO capabilities of dealing with problems of high dimensionality. We consider both ProFQI and ProDQN (Section 4.2), comparing their performance with their regular counterparts. To handle the complexity of the input space of the considered problem, we leverage neural network regression to model our PBO. We consider an offline setting, where we use ProFQI on car-on-hill (Ernst et al., 2005) and bicycle balancing Randlov & Alstrøm (1998), and an online setting, where we use ProDQN on bicycle balancing, and lunar lander (Brockman et al., 2016). We conduct this analysis to answer the following research question: does PBO enable moving toward the fixed point more effectively than the empirical Bellman operator? We provide a positive answer to this question, by focusing our analysis on the evaluation of performance w.r.t. Bellman iterations. Additional experimental details are in Appendix C.

6.1. Projected fitted Q-iteration

We initially evaluate ProFQI on the *car-on-hill* problem. As done in Ernst et al. (2005), we measure performance by generating roll-outs starting from different states on a grid of size 17×17, and accounting for the fact that the dataset $\mathcal{D}$ does not contain every starting state of the grid, by weighting the obtained performance from each starting state by the number of times it occurs in the dataset (see Appendix C.1.3). In Figure 8, we report the performance obtained with FQI and ProFQI for three different values of Bellman iterations K (black dashed vertical line). Again, for FQI, K is the number of regular iterations consisting of an application of the empirical Bellman operator and the projection step; for ProFQI, K is the number of applications of PBO that are used in the training loss (Equation 3). The iterations on the x-axis in Figure 8 are the regular iterations for FQI, and the applications of the trained PBO for ProFQI. Thus, the iterations on the right side of the line can only be reached with ProFQI. The purpose of this analysis is to evaluate the different quality of trajectories toward the fixed point obtained by the empirical Bellman operator in FQI, and the trained PBO in ProFQI, for the same amount of transition samples. We observe that ProFQI obtains better performance than FQI consistently. This evinces that PBO learns an accurate estimate of the true Bellman operator, which allows obtaining a satisfactory action-value function faster than the standard empirical Bellman operator used by



(a) $K = 2$. (b) $K = 3$. (c) $K = 4$.

Figure 6: ℓ_2-norm of the difference between the optimal action-value function parameters and the estimated ones on LQR. Results are averaged over 20 seeds.

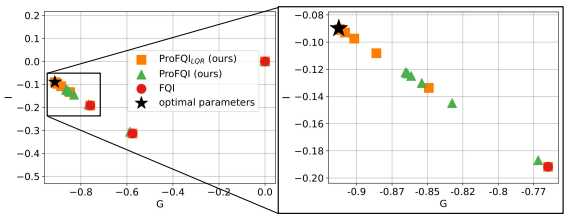


Figure 7: Iterated action-value function parameters in LQR.

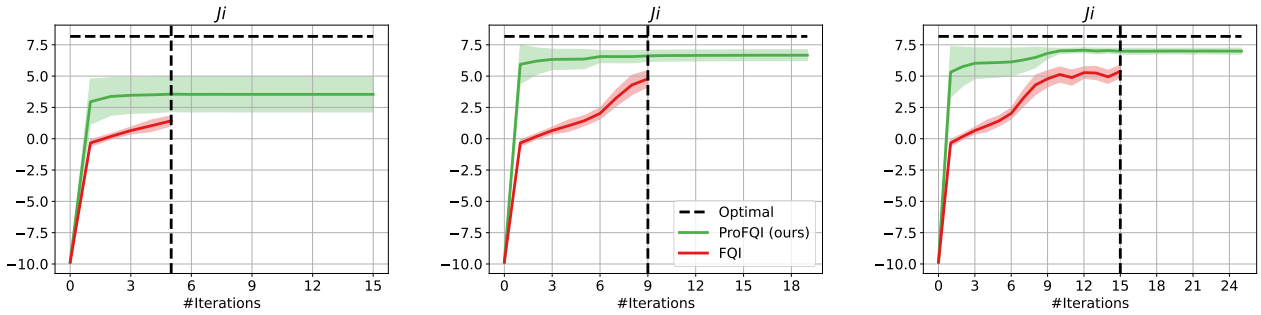
FQI. It is interesting to note that the performance of ProFQI does not grow for subsequent applications of PBO after the end of the training, showing that PBO is not able to get closer to the optimal action-value function due to limited training iterations, but conveniently remains stable. The benefit of PBO is further observable in the quality of the policy computed by FQI and ProFQI (Figure 10). After only 9 training iterations, ProFQI obtains a policy that is very close to the optimal one of car-on-hill (see the well-known shape of the optimal car-on-hill policy in Figure 10a (Ernst et al., 2005)), while FQI is significantly more inaccurate.

We perform another experiment on the more complex *bicycle balancing* problem (Randlov & Alstrøm, 1998). We set $K = 8$ and compute the discounted cumulative performance as done for car-on-hill. We observe that ProFQI achieves a significantly better performance than FQI in early iterations. In subsequent iterations, the performance has a small drop indicating that the learned PBO is not accurate enough to model further iterations. This could be due to a suboptimal choice of the architecture of the neural network of PBO, or a limited number of training iterations. Nevertheless, the performance remains higher than the one obtained by FQI.

6.2. Projected deep Q-network

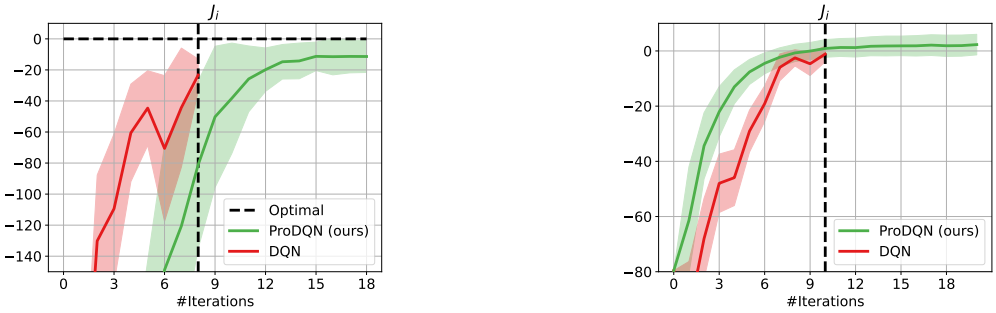
We also evaluate PBO in an online setting, using our ProDQN algorithm and comparing against DQN (Mnih et al., 2015). Again, we consider the *bicycle balancing* problem to draw a comparison with the results obtained in the offline setting, and the more complex *lunar lander* environment (Brockman et al., 2016). We set the number of Bellman iterations $K = 8$ for bicycle balancing and $K = 10$ for lunar lander. Similar to the offline setting, K is the number of updates of the target network for DQN, and the number of applications of PBO in the training loss (Equation 3) for ProDQN. Due to the need to collect new samples while learning a policy, training PBO in an online setting needs a slightly different treatment than the offline case. Recalling Algorithm 1, we point out that new samples are collected after each gradient descent step, by using the action-value function obtained after K applications of the current PBO. We find this choice to work well in practice; however, we can envision multiple other possibilities for effective exploration strategies based on PBO, that we postpone to future works. Figure 9 shows the discounted cumulative reward collected by DQN and ProDQN on both bicycle balancing and lunar lander, for different numbers of iterations. For DQN, each iteration corresponds to an update of the target network, while for ProDQN it indicates an application of the trained PBO. Regarding the bicycle balancing problem, we observe a substantially different behavior than the one of ProFQI in Figure 11. While ProFQI is able to outperform FQI in early iterations, ProDQN lags behind DQN. Moreover, while ProFQI exhibits a drop in performance for further iterations, ProDQN shows an increase in performance that eventually outperforms DQN. This behavior is interesting since it shows that, although slower, in this case PBO moves toward the fixed point more robustly than the empirical Bellman operator.

We conduct a similar evaluation on lunar lander. ProDQN outperforms DQN since early iterations and maintains stable performance, with a slight increase after K iterations. The achieved performance is far from optimal, but lunar lander is arguably a complex problem, where high-dimensionality



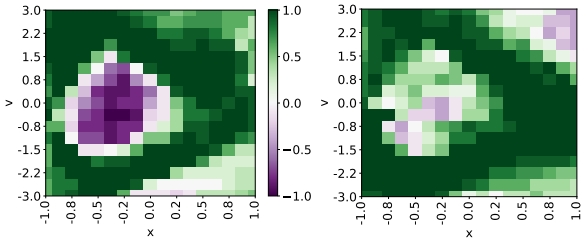
(a) $K = 5$. (b) $K = 9$. (c) $K = 15$.

Figure 8: Discounted cumulative reward on car-on-hill averaged over 20 seeds with 95% confidence intervals.



(a) Bicycle balancing. (b) Lunar lander.

Figure 9: Discounted cumulative reward on bicycle and lunar lander averaged over 40 seeds with 95% confidence intervals.



(a) ProFQI (ours). (b) FQI.

Figure 10: Policies on car-on-hill using $K = 9$ and 9 application of PBO.

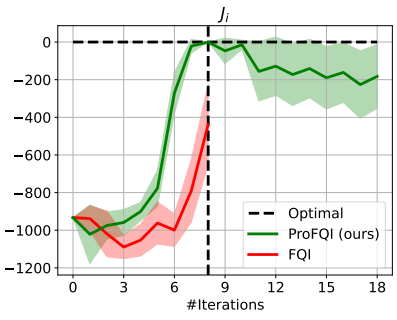


Figure 11: Discounted cumulative reward on bicycle balancing. Results are averaged over 20 seeds and 95% confidence intervals are shown.

and exploration pose significant challenges. These results, achieved with a limited number of action-value parameters to make PBO feasible, provide evidence of the effectiveness of PBO in deep RL problems. Further extensions, that we consider beyond the scope of this work, could leverage deep learning techniques to scale PBO to problems needing many action-value function parameters, e.g., Atari (Bellemare et al., 2013) or MuJoCo (Todorov et al., 2012).

7. Discussion and conclusion

We introduced the novel idea of an operator that directly maps parameters of action-value functions to others, as opposed to the regular Bellman operator that requires costly projections steps onto the space of action-value functions. This operator, which we call projected Bellman operator (PBO), generates a sequence of parameters that can progressively approach the ones of the optimal action-value function. One limitation of our method in its current form is that, given that the size of input and output spaces of PBO depends on the number of parameters of the action-value function, it is challenging to scale to problems that learn action-value functions with deep neural networks with millions of parameters (Mnih et al., 2015).

Acknowledgments

Funded by Hessian.ai through the Connectom Fund on Lifelong Explainable Robot Learning and the project 'The Third Wave of Artificial Intelligence – 3AI' by the Ministry for Science and Arts of the state of Hessen. We thank Daniel Palenicek and Tim Schneider for their support while performing our experiments on the IAS group's computing cluster at TU Darmstadt.

References

Agarwal, A., Jiang, N., Kakade, S. M., and Sun, W. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, pp. 10–4, 2019.

Agarwal, A., Kakade, S., Krishnamurthy, A., and Sun, W. Flambe: Structural complexity and representation learning of low rank mdps. *Advances in neural information processing systems*, 33:20095–20107, 2020.

Asadi, K. and Littman, M. L. An alternative softmax operator for reinforcement learning. In *International Conference on Machine Learning*, pp. 243–252. PMLR, 2017.

Bas-Serrano, J., Curi, S., Krause, A., and Neu, G. Logistic q-learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3610–3618. PMLR, 2021.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P., and Munos, R. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Bellemare, M. G., Dabney, W., and Rowland, M. *Distributional Reinforcement Learning*. MIT Press, 2023. http://www.distributional-rl.org.

Bellman, R. Dynamic programming. *Science*, 153(3731): 34–37, 1966.

Belousov, B. and Peters, J. Entropic regularization of markov decision processes. *Entropy*, 21(7):674, 2019.

Bertsekas, D. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

Bertsekas, D. P. *Dynamic Programming and Optimal Control 4 th Edition , Volume II*. Athena Scientific, 2015.

Bradtke, S. Reinforcement learning applied to linear quadratic regulation. *Advances in neural information processing systems*, 5, 1992.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Chen, T. and Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *JMLR*, 6:503–556, dec 2005. ISSN 1532-4435.

Farahmand, A.-m. Regularization in reinforcement learning. 2011.

Fellows, M., Hartikainen, K., and Whiteson, S. Bayesian bellman operators. *Advances in Neural Information Processing Systems*, 34:13641–13656, 2021.

Geist, M., Scherrer, B., and Pietquin, O. A theory of regularized markov decision processes. In *International Conference on Machine Learning*, pp. 2160–2169. PMLR, 2019.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pp. 1352–1361. PMLR, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kissas, G., Seidman, J. H., Guilhoto, L. F., Preciado, V. M., Pappas, G. J., and Perdikaris, P. Learning operators with coupled attention. *Journal of Machine Learning Research*, 23(215):1–63, 2022.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

Lagoudakis, M. G. and Parr, R. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4: 1107–1149, 2003.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Micchelli, C. A. and Pontil, M. On learning vector-valued functions. *Neural computation*, 17(1):177–204, 2005.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Munos, R. Error bounds for approximate policy iteration. In *ICML*, volume 3, pp. 560–567, 2003.

Munos, R. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, pp. 1006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

Munos, R. and Szepesvári, C. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9 (5), 2008.

Neu, G., Jonsson, A., and Gómez, V. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

Pang, B. and Jiang, Z.-P. Robust reinforcement learning: A case study in linear quadratic regulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9303–9311, 2021.

Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Randlov, J. and Alstrøm, P. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 463–471, 01 1998.

Sekhari, A., Dann, C., Mohri, M., Mansour, Y., and Sridharan, K. Agnostic reinforcement learning with low-rank mdps and rich observations. *Advances in Neural Information Processing Systems*, 34:19033–19045, 2021.

Song, Z., Parr, R., and Carin, L. Revisiting the softmax bellman operator: New benefits and new perspective. In *International conference on machine learning*, pp. 5916–5925. PMLR, 2019.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Tosatto, S., D'Eramo, C., Pajarinen, J., Restelli, M., and Peters, J. Exploration driven by an optimistic bellman equation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Watkins, C. J. C. H. Learning from delayed rewards. 1989.

A. Projected Bellman operator in low-rank Markov decision processes

Low-rank MDPs is a class of problems with two feature maps $\boldsymbol{\sigma} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and $\boldsymbol{\mu} : \mathcal{S} \to \mathbb{R}^d$, such that $\mathcal{P}(s'|s,a) = \langle \boldsymbol{\sigma}(s,a), \boldsymbol{\mu}(s') \rangle$ and $\mathcal{R}(s,a) = \langle \boldsymbol{\sigma}(s,a), \boldsymbol{\theta} \rangle$, for all $(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ and for $\boldsymbol{\theta} \in \mathbb{R}^d$ (Agarwal et al., 2020; Sekhari et al., 2021). We assume, without loss of generality, that for all $(s,a) \in \mathcal{S} \times \mathcal{A}$, $\|\boldsymbol{\sigma}(s,a)\|_1 \leq 1$ and $\max\{\|\boldsymbol{\mu}(s)\|_1, \|\boldsymbol{\theta}\|_1\} \leq \sqrt{d}$. The space of action-value functions is set as the space of linear functions in the parameters, i.e., $\mathcal{Q}_\Omega = \{\langle \boldsymbol{\sigma}(\cdot,\cdot), \boldsymbol{\omega} \rangle | \boldsymbol{\omega} \in \mathbb{R}^d\}$.

Proposition A.1. *In case of continuous state and action spaces and for $\boldsymbol{\omega} \in \mathbb{R}^d$, the PBO is*

$$\Lambda^*(\boldsymbol{\omega}) = \boldsymbol{\theta} + \gamma \int_{\mathcal{S}} \max_{a' \in \mathcal{A}} \langle \boldsymbol{\sigma}(s',a') | \boldsymbol{\omega} \rangle \boldsymbol{\mu}(s') \mathrm{d}s'. \tag{8}$$

The closed-form is again a γ-contraction mapping assuming that the MDP has a latent variable representation.

B. Proofs

B.1. Closed-form of PBO for a finite MDP

Proof. We compute the optimal Bellman iteration over a table $Q \in \mathbb{R}^{N \cdot M}$

$$\begin{aligned} \Gamma^* Q(s,a) &= r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} [Q(s',a')] \right] \\ &= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \left[\max_{a' \in \mathcal{A}} [Q(s',a')] \right] \\ &= \left(R + \gamma P \max_{a' \in \mathcal{A}} Q(\cdot, a') \right)(s,a) \end{aligned} \tag{9}$$

where $P \in \mathbb{R}^{N \cdot M \times N}$ is the transition probability matrix of the environment. From these equations, the operator $Q \mapsto R + \gamma P \max_{a' \in \mathcal{A}} Q(\cdot, a')$, evaluated on the objective function from the definition of PBO 2, yields zero error. This means that we have found the PBO in closed-form. □

B.2. Closed-form of PBO for LQR MDP

Proof. We assume that the distribution over the samples is a discrete uniform distribution over $\mathcal{S} \times \mathcal{A}$ centered on zero in both directions. With this assumption, the optimization problem 2 is equivalent to:

$$\arg\min_{\Lambda : \Omega \to \Omega} \mathbb{E}_{\boldsymbol{\omega} \sim \nu} \left[\sum_{(s,a) \in \bar{\mathcal{S}} \times \bar{\mathcal{A}}} \left(\Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Q_{\Lambda(\boldsymbol{\omega})}(s,a) \right)^2 \right]$$

where $\bar{\mathcal{S}} \times \bar{\mathcal{A}}$ is the set of all possible state-action pairs that can be drawn by the distribution of samples.

Let Λ be an operator on $\Omega = \mathbb{R}^2$, $\boldsymbol{\omega} = (G, I) \in \Omega$ and $(s,a) \in \bar{\mathcal{S}} \times \bar{\mathcal{A}}$. We note the first and second component of $\Lambda(\omega)$, $\Lambda_G(\omega)$ and $\Lambda_I(\omega)$, we have:

$$\Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Q_{\Lambda(\boldsymbol{\omega})}(s,a) = \begin{bmatrix} -s^2 & -2sa \end{bmatrix} \begin{bmatrix} \Lambda_G(\omega) \\ \Lambda_I(\omega) \end{bmatrix} + \Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Ma^2.$$

We note $Z(s,a) = \begin{bmatrix} -s^2 & -2sa \end{bmatrix}$, $X = \begin{bmatrix} \Lambda_G(\omega) & \Lambda_I(\omega) \end{bmatrix}^T$ and $b(s,a) = \Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Ma^2$. By summing over the samples we get:

$$\sum_{(s,a) \in \bar{\mathcal{S}} \times \bar{\mathcal{A}}} \left(\Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Q_{\Lambda(\boldsymbol{\omega})}(s,a) \right)^2 = \sum_{(s,a) \in \bar{\mathcal{S}} \times \bar{\mathcal{A}}} \left(Z(s,a)X + b(s,a) \right)^2 = ||ZX + b||_2^2$$

where $Z = \begin{bmatrix} \dots \\ Z(s,a) \\ \dots \end{bmatrix}$ and $b = \begin{bmatrix} \dots & b(s,a) & \dots \end{bmatrix}^T$.

Minimizing $||ZX + b||_2^2$ over $X \in \Omega$ will bring us to the minimizer of the optimization problem for any parameter

distribution ν. To minimize this quantity, we need to investigate the matrix

$$Z^T Z = \begin{bmatrix} \sum_s s^4 & 2\sum_{s,a} s^3 a \\ 2\sum_s s^3 a & 2\sum_{s,a} s^2 a^2 \end{bmatrix} = \begin{bmatrix} \sum_s s^4 & 0 \\ 0 & \sum_{s,a} s^2 a^2 \end{bmatrix}.$$

The last equality comes from the fact that the sample distribution is symmetric along $\bar{\mathcal{S}}$ and $\bar{\mathcal{A}}$. $Z^T Z$ is positive definite so $\min_{X\in\Omega} ||ZX + b||_2^2$ has a unique minimizer: $\Lambda^*(\omega) = -(Z^T Z)^{-1} Z^T b$.

Let us now rewrite b. The optimal Bellman iteration over $Q_{\boldsymbol{\omega}}$ is $\Gamma^* Q_{\boldsymbol{\omega}}(s,a) = r(s,a) + \max_{a'} Q_{\boldsymbol{\omega}}(s',a')$. M is chosen negative so that the function $a' \mapsto Q_{\boldsymbol{\omega}}(s',a') = G \cdot s'^2 + 2I \cdot s'a' + M \cdot a'^2$ has a unique maximizer of equation $-I/M \cdot s'$. This makes

$$\max_{a'} Q_{\boldsymbol{\omega}}(s',a') = Q_{\boldsymbol{\omega}}(s', -\frac{I}{M} \cdot s') = G \cdot s'^2 - 2\frac{I^2}{M} \cdot s'^2 + \frac{I^2}{M} \cdot s'^2 = (G - \frac{I^2}{M}) \cdot s'^2.$$

By inserting it into the Bellman equation, we get

$$\begin{aligned} b(s,a) &= \Gamma^* Q_{\boldsymbol{\omega}}(s,a) - Ma^2 \\ &= r(s,a) + (G - \frac{I^2}{M}) \cdot s'^2 - Ma^2 \\ &= Q \cdot s^2 + 2S \cdot sa + R \cdot a^2 + (G - \frac{I^2}{M}) \cdot s'^2 - Ma^2 \\ &= \left(Q + A^2(G - \frac{I^2}{M})\right) \cdot s^2 + 2\left(S + AB(G - \frac{I^2}{M})\right) \cdot sa + \left(R + B^2(G - \frac{I^2}{M}) - M\right) \cdot a^2 \\ &= \begin{bmatrix} s^2 & 2sa & a^2 \end{bmatrix} \cdot \begin{bmatrix} Q + A^2(G - \frac{I^2}{M}) - G \\ S + AB(G - \frac{I^2}{M}) - I \\ R + B^2(G - \frac{I^2}{M}) - M \end{bmatrix}. \end{aligned}$$

This means that

$$\Lambda^*(\omega) = -(Z^T Z)^{-1} Z^T J \begin{bmatrix} Q + A^2(G - \frac{I^2}{M}) \\ S + AB(G - \frac{I^2}{M}) \\ R + B^2(G - \frac{I^2}{M}) - M \end{bmatrix}$$

where $J = \begin{bmatrix} \cdots & & \\ s^2 & 2sa & a^2 \\ & \cdots & \end{bmatrix}$.

From the fact that $Z^T J = \begin{bmatrix} -\sum_s s^4 & 0 & 0 \\ 0 & \sum_{s,a} s^2 a^2 & 0 \end{bmatrix}$, we have $-(Z^T Z)^{-1} Z^T J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, thus $\Lambda^*(\omega) = \begin{bmatrix} Q + A^2(G - \frac{I^2}{M}) \\ S + AB(G - \frac{I^2}{M}) \end{bmatrix}$.

□

Remarks With the assumption on the distribution of sample, the PBO can also be understood in a geometrical way. It projects the parameters along a line of direction vector $\begin{bmatrix} A^2 & AB \end{bmatrix}^T$ with an offset $\begin{bmatrix} Q & S \end{bmatrix}^T$. The iterations correspond to a non-linear transformation of the coefficient $(G - {}^{I^2}/_M)$ in front of the direction vector. This also means that the fixed-point, i.e. the optimal parameters are also on this line.

B.3. Closed-form of PBO for a low-rank MDP

Proof. The proof considers continuous unbounded state action spaces. For $Q_{\boldsymbol{\omega}} \in \mathcal{Q}_\Omega$, $\boldsymbol{\omega}$ the vector representing $Q_{\boldsymbol{\omega}}$, $\max_{a\in\mathcal{A}} Q_{\boldsymbol{\omega}}(s,a)$ is well defined (here $\max$ might not be attained, it should be interpreted as a supremum). We have $|Q_{\boldsymbol{\omega}}(s,a)| = |\langle \boldsymbol{\sigma}(s,a) | \boldsymbol{\omega} \rangle| \leq ||\boldsymbol{\sigma}(s,a)|| \cdot ||\boldsymbol{\omega}|| \leq ||\boldsymbol{\omega}||$, thus $\max_{a\in\mathcal{A}} Q_{\boldsymbol{\omega}}(s,a) < \infty$. Then, we write the optimal Bellman

iteration on the function $Q_{\boldsymbol{\omega}}$ as

$$\begin{aligned}
\Gamma^* Q_{\boldsymbol{\omega}}(s,a) &= r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_{\boldsymbol{\omega}}(s',a') \right] \\
&= r(s,a) + \gamma \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | \boldsymbol{\omega} \rangle p(s'|s,a) \mathrm{d}s' \\
&= \langle \boldsymbol{\sigma}(s,a) | \boldsymbol{\theta} \rangle + \gamma \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | \boldsymbol{\omega} \rangle \langle \boldsymbol{\sigma}(s,a) | \boldsymbol{\mu}(s') \rangle \mathrm{d}s' \text{ from the definition of the transition} \\
&\text{probabilities.} \\
&= \langle \boldsymbol{\sigma}(s,a) | \boldsymbol{\theta} + \gamma \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | \boldsymbol{\omega} \rangle \boldsymbol{\mu}(s') \mathrm{d}s' \rangle \text{ from the linear of the scalar product.}
\end{aligned}$$

This derivation shows that the operator $\boldsymbol{\omega} \mapsto \boldsymbol{\theta} + \gamma \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | \boldsymbol{\omega} \rangle \boldsymbol{\mu}(s') \mathrm{d}s'$ minimizes the objective function presented in the definition of PBO 2 since its yields zero error. This operator is the PBO for a low-rank MDP. □

B.4. PBO is a γ-contraction mapping for a low-rank MDP

Proof. We now assume that the MDP has a latent variable representation. This proof was inspired by the proof of the contraction property of the Bellman Operator in Bertsekas (2015). Considering $\boldsymbol{\omega}, \boldsymbol{\omega}' \in \mathbb{R}^d$, we have

$$\begin{aligned}
||\Gamma_p^*(\boldsymbol{\omega}) - \Gamma_p^*(\boldsymbol{\omega}')||_\infty &= \gamma || \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | (\boldsymbol{\omega} - \boldsymbol{\omega}') \rangle \boldsymbol{\mu}(s') \mathrm{d}s' ||_\infty \\
&= \gamma \max_{i \in \{1,\dots,d\}} \left| \int_{s'} \max_{a'} \langle \boldsymbol{\sigma}(s',a') | (\boldsymbol{\omega} - \boldsymbol{\omega}') \rangle \boldsymbol{\mu}(s')_i \mathrm{d}s' \right| \\
&\leq \gamma \max_{i \in \{1,\dots,d\}} \int_{s'} \max_{a'} |\langle \boldsymbol{\sigma}(s',a') | (\boldsymbol{\omega} - \boldsymbol{\omega}') \rangle| \, \boldsymbol{\mu}(s')_i \mathrm{d}s' \text{ since } \boldsymbol{\mu}(\cdot) \text{ is positive.} \\
&\leq \gamma \max_{i \in \{1,\dots,d\}} \int_{s'} \max_{a'} \sum_{j \in \{1,\dots,d\}} \boldsymbol{\sigma}(s',a')_j |(\boldsymbol{\omega} - \boldsymbol{\omega}')_j| \boldsymbol{\mu}(s')_i \mathrm{d}s' \text{ since } \boldsymbol{\sigma}(\cdot,\cdot) \text{ are positive.} \\
&\leq \gamma \max_{i \in \{1,\dots,d\}} \int_{s'} \max_{a'} \sum_{j \in \{1,\dots,d\}} \boldsymbol{\sigma}(s',a')_j \boldsymbol{\mu}(s')_i \mathrm{d}s' \cdot ||\boldsymbol{\omega} - \boldsymbol{\omega}'||_\infty \\
&\leq \gamma \max_{i \in \{1,\dots,d\}} \int_{s'} \boldsymbol{\mu}(s')_i \mathrm{d}s' \cdot ||\boldsymbol{\omega} - \boldsymbol{\omega}'||_\infty \text{ since } \boldsymbol{\sigma}(\cdot,\cdot) \text{ are probability distributions.} \\
&\leq \gamma ||\boldsymbol{\omega} - \boldsymbol{\omega}'||_\infty \text{ since } \boldsymbol{\mu}(\cdot)_i \text{ is a probability distribution for all } i.
\end{aligned}$$

□

C. Details of the empirical analysis

We provide details of the experimental setting. Table 1 and 2 summarize the values of all parameters appearing in the experiments. FQI, DQN, ProFQI and ProDQN use Adam optimizer (Kingma & Ba, 2015) with a linear annealing learning rate. For FQI, the optimizer is reset at each iteration. The set of parameters $\mathcal{W}$ is generated by sampling from a truncated normal distribution. When we use action-value functions with a neural network that has more than one hidden layer, the last one is initialized with zeros. This way, the output of a neural network parameterized by any element of $\mathcal{W}$ is zero which makes the reward easier to learn. Among the parameters in $\mathcal{W}$, one is chosen to be the initial parameters used by FQI or DQN. For all the methods, the metrics have been constructed starting from the initial parameters, this is why, all the plots showing the performances share the same J_0. As Table 1 and 2 show, the initial parameters of PBOs are always taken small enough so that applying PBO multiple times do not lead to diverging outputs.

Table 1: Summary of all parameters used in the offline experiments.

		Chain-walk	LQR	Car-on-hill	Bicycle
horizon		$+\infty$	$+\infty$	100	50.000
γ		0.9	1	0.95	0.99
$\#\mathcal{D}$		400	121	5.500	70.000
batch size on $\mathcal{D}$		20	121	500	1.000
FQI	#fitting steps	400	800	1.200	1.200
	#patience steps	100	100	30	7
	starting learning rate	10^{-2}	10^{-2}	10^{-3}	5×10^{-3}
	ending learning rate	10^{-5}	10^{-5}	5×10^{-7}	10^{-4}
ProFQI	$\#\mathcal{W}$	100	5	30	50
	batch size on $\mathcal{W}$	100	5	30	25
	#epochs	1.000	1.000	1.000	500
	#training steps	5	4	10	20
	starting learning rate	10^{-2}	10^{-2}	10^{-3}	10^{-4}
	ending learning rate	10^{-7}	10^{-5}	5×10^{-7}	10^{-7}
	initial PBO's parameters std	5×10^{-6}	5×10^{-6}	5×10^{-7}	5×10^{-7}

C.1. Offline experiments

C.1.1. Chain-walk

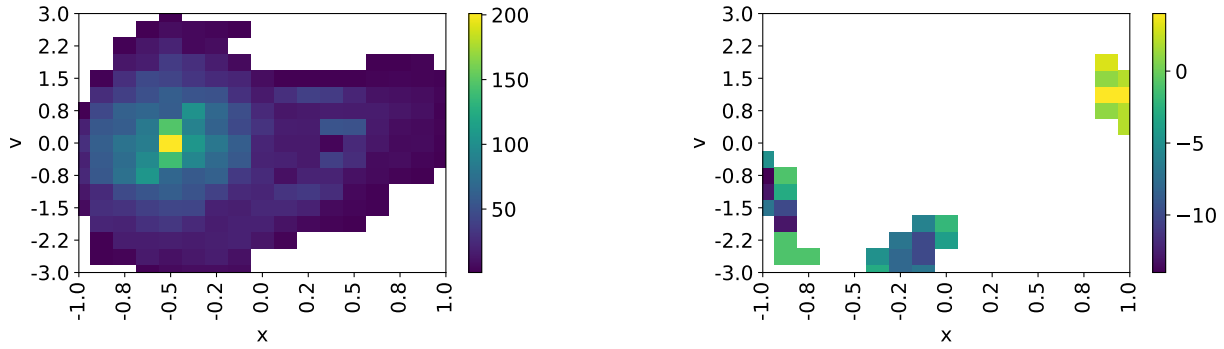
We consider all the possible state-action pairs 10 times as the initial dataset of samples $\mathcal{D}$. These 10 repetitions help the algorithms to grasp the randomness of the environment. The optimal action-value function Q^* is computed with dynamic programming. The ℓ^2-norm is computed by taking the norm of the vector representing the action-value function. In Figure 4, we only report the mean value, since the standard deviations over the seeds are negligible.

C.1.2. Linear quadratic regulator

The dynamics of the system is $\mathcal{P}(s,a) = -0.46s + 0.54a$ and the reward function is $\mathcal{R}(s,a) = -0.73s^2 - 0.63sa - 0.93a^2$. M is chosen to be equal to -1.20 The set $\mathcal{D}$ is collected on a mesh of size 11 by 11 on the state-action space going from -4 to 4 in both directions, which means that samples belong to $[-4,4] \times [-4,4] \subset \mathcal{S} \times \mathcal{A}$. We choose to parameterize the value functions in a quadratic way, thus allowing us to compute the maximum in closed-form. However, we assume that the algorithms do not have this knowledge since it is not the case in more general settings. For this reason, we discretize the action space in a set of 200 actions going from -8 to 8. The architecture of the parameterized PBO trained with ProFQI is a fully connected network composed of one hidden layer having 8 neurons. Rectified linear unit (ReLU) is used as activation functions. In Figure 6, we only report the mean value, since the standard deviations over the seeds are negligible.

C.1.3. Car-on-hill

The agent chooses between 2 actions: `left` or `right`. The state space is 2-dimensional: position in $[-1,1]$ and velocity $[-3,3]$. If the agent succeeds to bring the car up the hill – at position greater than 1 and velocity in between -3 and 3 – then the reward is 1, if the agent exceeds the state space, the reward is -1; otherwise, the reward is 0.
As our ProFQI is an offline algorithm, we need to make sure that all the necessary exploration has been done in the dataset of samples. For that reason, we first consider a uniform sampling policy to collect episodes starting from the lowest point in the map ($[-0.5, 0]$) with an horizon of 100. This sampling process is stopped when 4.500 samples are gathered. To get more samples with positive reward, we sample new episodes starting from a state located randomly between $[0.1, 1.3]$ and $[0.5, 0.38]$ with a uniform policy as well. In total, 5.500 samples are collected. The sample and reward distributions over the state space is shown in Figure 12. The action-value functions are parameterized with one hidden layer of 30 neurons with ReLU as activation functions. The architecture of the parameterized PBO has 4 hidden layers of 302 (2 times the number of parameters of the action-value functions) neurons each with ReLU as activation functions. To help the training, the value functions are taking actions in $\{-1, 1\}$ instead of the usual $\{0, 1\}$. Given that the policies, the reward, and the dynamics, are deterministic, we perform only one simulation to generate Figure 8.



(a) Sample distribution.

(b) Reward distribution.

Figure 12: Composition of the dataset of samples $\mathcal{D}$ on car-on-hill.

C.1.4. BICYCLE

We consider the bicycle problem, as described in Randlov & Alstrøm (1998). The state space is composed of 4 dimensions: $(\omega, \dot{\omega}, \theta, \dot{\theta})$ where ω is the angle between the floor and the bike, and θ is the angle between the handle bar and the perpendicular axis to the bike. The goal is to ride a bicycle for 500 seconds (50.000 steps). The agent can apply a torque $T \in \{-2, 0, 2\}$ to the handle bar to make it rotate. The agent can also move its center of gravity in the direction $d \in \{-0.02, 0, 0.02\}$ perpendicular to the bike. As in Lagoudakis & Parr (2003), the agent chooses between applying a torque or moving its center of gravity, resulting in 5 actions instead of 9. Usually, a uniform noise in the interval $[-0.02, 0.02]$ is added to d. For the purpose of this work, we reduce the number of samples by making the magnitude of the noise 10 times smaller. A reward of -1 is given when the bike falls down, i.e., $|\omega| > 12°$. We use reward shaping to have more informative samples, and we add a reward proportional to the change in ω, i.e., $10^4(|\omega_t| - |\omega_{t+1}|)$, as in Lagoudakis & Parr (2003). The dataset of samples is composed of 3.500 episodes starting from a position close to $(0, 0, 0, 0)$ and cut after 20 steps (Lagoudakis & Parr, 2003). The action-value functions are parameterized with one hidden layer of 30 neurons with ReLU activations. The architecture of the parameterized PBO is composed of 3 hidden layers of 302 neurons (2 times the number of parameters of the action-value functions) each with ReLU functions as activation functions. 100 simulations are done for each seed shown in Figure 11, all of them starting for the state $(0, 0, 0, 0)$, i.e., the bicycle standing straight.

C.2. Online experiments

C.2.1. BICYCLE

We also consider the bicycle problem in an online setting. The architecture of the action-value functions and the parameterized PBO remain the same. During sampling, we only leave the algorithms 20 steps to explore before ending the episode like in the offline setting. This is done to avoid exploring regions in the state space that are not useful for solving the environment. 100 simulations are done for each seed shown in Figure 9a.

C.2.2. LUNAR LANDER

Lunar lander, introduced in (Brockman et al., 2016), is an environment in which the goal is to make a lunar module land at a specific location while behaving safely for the crew and the rocket. The state space is composed of 8 dimensions: the position of the rocket, the linear velocities, the angle with the horizon, the angular velocity and two booleans for each leg being activated when they touch the ground. The action space consists of 4 actions: fire the main engine, fire the left engine, fire the right engine, do nothing. The action-value functions are parameterized with 2 hidden layers of 30 neurons each with ReLU as activation functions. The architecture of the parameterized PBO is composed of 4 hidden layers of 2522 neurons (2 times the number of parameters of the action-value functions) each with ReLU functions as activation functions. 100 simulations are done for each seed shown in Figure 9a each of them starting from a random initial position.

Table 2: Summary of all parameters used in the online experiments.

		Bicycle	Lunar Lander
horizon		50.000	1.000
γ		0.99	0.99
#$\mathcal{D}$		10.000	1.000
$\max$ #$\mathcal{D}$		10.000	20.000
batch size on $\mathcal{D}$		500	500
#steps per update		2	2
starting ϵ		1	1
ending ϵ		10^{-2}	10^{-2}
DQN	#fitting steps	6.000	6.000
	starting learning rate	10^{-4}	10^{-3}
	ending learning rate	10^{-6}	10^{-5}
ProDQN	#$\mathcal{W}$	30	30
	batch size on $\mathcal{W}$	30	15
	#epochs	4.000	3.000
	#training steps	25	25
	starting learning rate	10^{-5}	10^{-5}
	ending learning rate	10^{-7}	5×10^{-7}
	initial PBO's parameters std	5×10^{-7}	5×10^{-7}