Parameterized Projected Bellman Operator

Théo Vincent1,2*, Alberto Maria Metelli3, Boris Belousov1, Jan Peters1,2,4,5, Marcello Restelli1, Carlo D’Eramo2,4,6
1German Research Center for AI (DFKI), Germany
2Department of Computer Science, TU Darmstadt, Darmstadt, Germany
3Department of Electronics, Computer Science, and Bioengineering, Politecnico di Milano, Milano, Italy
4Hessian.ai, Germany
5Centre for Cognitive Science, TU Darmstadt, Darmstadt, Germany
6Center for Artificial Intelligence and Data Science, University of Würzburg, Würzburg, Germany

Abstract
Approximate value iteration (AVI) is a family of algorithms for reinforcement learning (RL) that aims to obtain an approximation of the optimal value function. Generally, AVI algorithms implement an iterated procedure where each step consists of (i) an application of the Bellman operator and (ii) a projection step into a considered function space. Notoriously, the Bellman operator leverages transition samples, which strongly determine its behavior, as uninformative samples can result in negligible updates or long detours, whose detrimental effects are further exacerbated by the computationally intensive projection step. To address these issues, we propose a novel alternative approach based on learning an approximate version of the Bellman operator rather than estimating it through samples as in AVI approaches. This way, we are able to (i) generalize across transition samples and (ii) avoid the computationally intensive projection step. For this reason, we call our novel operator projected Bellman operator (PBO). We formulate an optimization problem to learn PBO for generic sequential decision-making problems, and we theoretically analyze its properties in two representative classes of RL problems. Furthermore, we theoretically study our approach under the lens of AVI and devise algorithmic implementations to learn PBO in offline and online settings by leveraging neural network parameterizations. Finally, we empirically showcase the benefits of PBO w.r.t. the regular Bellman operator on several RL problems.

Introduction
Value-based reinforcement learning (RL) is a popular class of algorithms for solving sequential decision-making problems with unknown dynamics (Sutton and 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 and Szepesvári 2008). Moreover, in AVI approaches, costly function approximation steps are needed to project the output of the Bellman operator back to the considered value function space.

In this paper, we introduce the projected Bellman operator (PBO), which consists of a function \( \Lambda : \Omega \rightarrow \Omega \) defined on the parameters \( \omega \in \Omega \) of the value function approximator \( Q_\omega \). Contrary to the standard (empirical) Bellman operator \( \Gamma \), which acts on the value functions \( Q_k \) to compute targets that are then projected to obtain \( Q_{k+1} \), our PBO \( \Lambda \) acts on the parameters of the value function to directly compute updated parameters \( \omega_{k+1} = \Lambda(\omega_k) \) (Figure 1). The advantages of our approach are twofold: (i) PBO is applicable for an arbitrary number of iterations without using further samples, and (ii) the output of PBO always belongs to the considered value function space as visualized in Figure 2, thus avoiding the costly projection step which is required when using the Bellman operator coupled with function approximation. We show how to estimate PBO from transition samples by leveraging a parametric approximation which we call parameterized PBO, and we devise two algorithms for offline and online RL to learn it. Starting from initial parameters \( \omega_0 \), AVI approaches obtain consecutive approximations of the value function \( Q_\omega \), 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 \( \omega_0 \), PBO can produce a chain of value function parameters of arbitrary length (as shown with the blue lines in Figure 2a) without requiring further samples. This means an

\[
\omega_k \xrightarrow{\Lambda} \omega_{k+1} \xrightarrow{\Gamma} \omega_{k+2} \xrightarrow{\Pi_{proj}} Q_{\omega_{k+1}}
\]

Figure 1: PBO \( \Lambda \) (left) operates on value function parameters as opposed to AVI (right), that uses the empirical Bellman operator \( \Gamma \) followed by the projection operator \( \Pi_{proj} \).
accurate approximation of PBO can compute optimal value function parameters starting from any initial parameters in the chosen space without requiring additional samples.

**Related Work**

Several papers in the literature proposed variants of the standard Bellman operator to induce certain behaviors. We review these approaches, noting that they all act on the space of action-value functions, thus needing a costly projection step onto the considered function space. Conversely, to the best of our knowledge, our work is the first attempt to obtain an alternative Bellman operator that avoids the projection step by directly acting on the parameters of value functions.

**Bellman operator variations.** Variants of the Bellman operator are widely studied for entropy-regularized MDPs (Neu, Jonsson, and Gómez 2017; Geist, Scherrer, and Pietquin 2019; Belousov and Peters 2019). The softmax (Haarnoja et al. 2017; Song, Parr, and Carin 2019), mellowmax (Asadi and 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 sub-optimal actions. The distributional Bellman operator (Bellemare, Dabney, and Munos 2017) enables operating on the whole return distribution instead of its expectation, i.e., the value function (Bellemare, Dabney, and Rowland 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, Hartikainen, and Whiteson 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, in contrast 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 and Pontil 2005) and parametric approaches for learning non-linear operators (Chen and Chen 1995), with a resurgence of recent contributions in deep learning. For example, Kovachki et al. (2021, 2023) 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.

**HyperNetworks.** Our approach shares similarities with HyperNetworks (Ha, Dai, and Le 2016; Sarafian, Keynan, and Kraus 2021; Beck et al. 2023). In the classification proposed in Chauhan et al. (2023), PBO distinguishes itself by taking as input the parameters generated at the previous forward pass. In this sense, PBO can be seen as a recurrent variant of HyperNetworks that generates a series of parameters representing the consecutive Bellman iterations.

**Preliminaries**

We consider discounted Markov decision processes (MDPs) defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a measurable state space, $\mathcal{A}$ is a finite, measurable action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ 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 policy $\pi : \mathcal{S} \to \mathcal{A}$ is a function mapping a state to an action, inducing a value function $V^\pi(s) \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t)) | S_0 = s \right]$ representing the expected cumulative discounted reward starting in state $s$ and following policy $\pi$ thereafter. Similarly, the action-value function $Q^\pi(s, a) \triangleq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) | S_0 = s, A_0 = a, A_k = \pi(S_k) \right]$ is the expected discounted cumulative reward executing action $a$ in state $s$, following policy $\pi$ thereafter. RL aims to find

---

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

**Figure 2:** Behavior of our PBO and AVI in the parametric space of value functions $\mathcal{Q}$. Here $Q^*_\omega$, $Q^*_{\omega^*}$, and $Q^*_{\Lambda^*}$ are respectively the optimal value function, its projection on the parametric space, and the fixed point of PBO. Contrary to the regular Bellman operator, PBO can be applied for an arbitrary number of steps (blue lines) without requiring additional samples.
an optimal policy \( \pi^* \) yielding the optimal value function
\( V^*(s) \triangleq \max_{a \in A} V^\pi(s) \) for every \( s \in S \) (Puterman 1990). The (optimal) Bellman operator \( \Gamma^* \) is a fundamental
tool in RL for obtaining optimal policies, defined as:

\[
(\Gamma^* Q)(s, a) \triangleq R(s, a) + \gamma \int_S \mathcal{P}(ds'|s, a) \max_{a' \in A} Q(s', a'),
\]

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

### Projected Bellman Operator

The application of the Bellman operator in RL requires tran-
sition samples (Equation 1) and a costly projection step onto
the used function space (Munos 2003, 2005; Munos and Szepesvári 2008). We are interested in obtaining an operator
that overcomes these issues while emulating the behavior of the
regular Bellman operator. Hence, we introduce the pro-
jected Bellman operator (PBO), which we define as follows.

**Definition 1.** Let \( Q_\Omega = \{Q_\omega : S \times A \to \mathbb{R} \mid \omega \in \Omega\} \) be a function
approximation space for the action-value function, induced by the parameter space \( \Omega \). A projected Bellman operator
(PBO) is a function \( \Lambda : \Omega \to \Omega \), such that

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

for state-action and parameter distributions \( \rho \) and \( \nu \).

Note that \( \Gamma^* \) is the regular optimal Bellman operator on
\( Q_\Omega \). Conversely, PBO is an operator \( \Lambda \) acting on action-
value function parameters \( \omega \in \Omega \). In other words, this de-
finite statement says that a PBO is the mapping \( \Omega \to \Omega \) that
closely emulates the behavior of the regular optimal Bellman
operator \( \Gamma^* \). Thus, by acting on the parameters \( \omega \) of
action-value functions \( Q_\omega \), PBO can be applied for an arbi-
trary number of steps starting from any initial parameteriza-
tions \( \omega_k \), without using additional transition samples. More-
over, being a mapping between parameters \( \omega_k \) to param-
ters \( \omega_{k+1} \), PBO does not require the costly projection step
needed by the regular Bellman operator.

### Learning Projected Bellman Operators

The PBO is unknown and has to be estimated from samples.
We propose to approximate PBO with a parameterized PBO
\( \Lambda_\phi \) differentiable w.r.t. its parameters \( \phi \in \Phi \), enabling the
use of gradient descent on a loss function.\(^2\) We do so by
formulating an empirical version of the problem (2) that can
be optimized via gradient descent by using a given dataset of
parameters \( \omega \in \mathcal{W} \sim \nu \) and transitions \((s, a, r, s') \in \mathcal{D} \sim \rho \). Crucially, we can leverage PBO to augment the dataset of

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

---

**Algorithm 1: Projected FQI & *Projected DQN*\(^*\)**

1: **Inputs:**
   - samples \( \mathcal{D} = \{(s_j, a_j, r, s_j')\}_{j=1}^J \);
   - parameters \( \mathcal{W} = \{\omega_j\}_{j=1}^J \);
   - #Bellman iterations \( K \);
   - initial parameters \( \phi \) of parameterized PBO \( \Lambda_\phi \);
   - #Epochs \( E \).

2: for \( e \in \{1, \ldots, E\} \) do
   3: \( \bar{\phi} = \phi \)
   4: for some training steps do
   5:   *Collect samples \( \mathcal{D}' \) with policy given by \( Q_{\Lambda_\phi^e}(\omega)^* \)
   6:   \( *\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'^* \)
   7:   Gradient descent over parameters \( \phi \) minimizing loss (3) or (4) using \( \mathcal{D}^* \) or a batch of \( \mathcal{D}^* \) and \( \mathcal{W} \).

8: end for

9: end for

10: **Return:** Parameters \( \phi \) of parameterized PBO \( \Lambda_\phi \)

---

parameters \( \omega \in \mathcal{W} \) with sequences generated by PBO at no cost of additional samples. The resulting loss is

\[
\mathcal{L}_{\Lambda_\phi} = \sum_{\omega \in \mathcal{W}} \mathbb{E}_{(s,a) \sim \rho, \omega \sim \nu} \left[ (\Gamma^* Q_{\omega}(s, a) - Q_{\Lambda_\phi}(\omega)(s, a))^2 \right],
\]

where \( K \) is an arbitrary number of optimal Bellman opera-
tor iterations. Note that for \( K = 1 \) we obtain the empirical
version of the optimization problem (2). An additional idea is to add a term that corresponds to an infinite number of
iterations, i.e., the fixed point,

\[
\mathcal{L}_{\Lambda_\phi^*} = \mathcal{L}_{\Lambda_\phi} + \sum_{(s,a) \in \mathcal{D}} \left( \Gamma Q_{\Lambda_\phi^*}(s, a) - Q_{\Lambda_\phi^*}(s, a) \right)^2
\]

where \( \Lambda_\phi^* \) is the fixed point of the parameterized PBO. Note
that the addition of the fixed-point term is only possible for
classes of parameterized PBOs where the fixed point can be
computed, and the result is differentiable w.r.t. the param-
ters \( \phi \), as we describe in the following section.

We can now devise two algorithms to learn PBO in both
offline and online RL. Our algorithms can be seen as vari-
ants of the offline algorithm fitted \( Q \)-iteration (FQI) (Ernst,
Geurts, and Wehenkel 2005) and online algorithm deep \( Q \)-
network (DQN) (Mnih et al. 2015); thus, we call them pro-
jected fitted \( Q \)-iteration (ProFQI) and projected deep \( Q \)-
network (ProDQN).

Algorithm 1 compactly describes both ProFQI and ProDQN, 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 execut-
ing a policy derived from the action-value function ob-
tained by applying the current approximation of PBO for
\( K \) times. For stability reasons, we perform gradient de-
scent steps only on the parameters \( \phi \) of \( Q_{\Lambda_\phi} \) in the loss

\[^*\text{For ease of presentation, we use PBO and parameterized PBO interchangeably whenever clear from the context.}\]
The PBO exists, and it is equal to the optimal Bellman operator. As a consequence of Theorem 2, the approximation error of FQI

\[
\|Q_T^k - Q^*\|_{\frac{1}{2}} \leq C_{K, \gamma, R} \min_{\pi \in \Pi} \|Q^\pi_k - Q^\pi_k\|_{\frac{1}{2}} + \frac{1}{\gamma} \log \left( \frac{1}{|A|} \right)
\]

This theorem shows that the distance between the value function and the optimal value function (left-hand side of Equation (5)) is upper-bounded by a quantity proportional to the approximation errors \(\|Q^\pi_k - Q^\pi_k\|_{\frac{1}{2}}\) at each iteration \(k\). As highlighted in Equation (5), at every iteration \(k\), the loss of FQI contains only one term of the sum. Conversely, the loss of ProFQI contains the entire sum of approximation errors from iteration \(k = 1\) to \(K\). This means that while FQI minimizes the approximation error for a single iteration at a time, ProFQI minimizes the approximation errors for multiple iterations, thus effectively reducing the upper bound on the approximation error at iteration \(K\).

In the following, we empirically showcase this theoretical advantage and we further analyze the properties of PBO for two representative classes of RL problems, namely (i) finite MDPs and (ii) linear quadratic regulators (Bradtke 1992; Pang and Jiang 2021). Proofs of the following results and additional analysis of PBO in low-rank MDPs (Agarwal et al. 2020; Sekhari et al. 2021) can be found in the appendix.

**Finite Markov Decision Processes**

Let us consider finite state and action spaces of cardinality \(N\) and \(M\), respectively, and a tabular setting with \(\Omega = R^N \times M\). 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 \(Q_\Omega\) and \(\Omega\), which allows us to write, for ease of notation, the parameters of the action-value function as \(Q\) instead of \(\omega\).

**Proposition 3.** The PBO exists, and it is equal to the optimal Bellman operator

\[
\Lambda^*(Q) = R + \gamma \max_{a \in A} Q(\cdot, a) \tag{6}
\]

Note that PBO for finite MDPs is a \(\gamma\)-contraction mapping for the \(L_\infty\)-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 \( K \in \{2, 5, 15\} \). For FQI, the \( K \) iterations are the regular iterations where the empirical Bellman operator is applied to the current approximation of the action-value function. For ProFQI, \( K \) is the number of iterations included in the PBO loss. 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 \( P \). 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 explain this behavior with the fact that when the linear approximation is inadequate for modeling PBO, adding the fixed point in the loss could harm the estimate.

![Figure 5: Test task: chain-walk from Munos (2003). The reward is 0 in all states except green states where it equals 1.](image)

### Linear Quadratic Regulation

Now, we consider the continuous MDPs class of linear quadratic regulator (LQR) with \( S = A = \mathbb{R} \). The transition model \( P(s, a) = As + Ba \) is deterministic and the reward function \( R(s, a) = Qs^2 + 2Sa + Ra^2 \) 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 \( Q_\Omega = \{(s, a) \mapsto Gs^2 + 2Is + M^2\} \) where \( M \) is a chosen constant, for visualization purposes.

**Proposition 4.** PBO exists, and for any \( \omega \in \mathbb{R}^2 \), its closed form is given by:\(^3\)

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

We leverage our theoretical analysis for LQR (Bradtke 1992; Pang and 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 the 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 solution 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

---

\(^3\)Under a mild assumption over the sample distribution, see in the Proofs section in the appendix.
Figure 6: $\ell_2$-norm of the difference between the optimal action-value function parameters and the estimated ones on LQR. Here $K$ is the number of iterations included in the training loss (3) of PBO, while $k$ is the number of PBO applications after training. Results are averaged over 20 seeds. Similar to Figure 4, PBO shows improved convergence for $k \geq K$ compared to FQI.

Figure 7: Behavior of FQI and ProFQI in the space of action-value function parameters $(G, I)$ for LQR. FQI is performed with 2 iterations, and ProFQI uses a PBO trained with $K = 2$, for the sake of fair comparison. Notably, ProFQI can leverage PBO by applying it for 8 iterations, being able to get closer to the optimal parameters (black star) than FQI.

Figure 8: Policies on car-on-hill using $K = 9$ and 9 application of PBO, averaged over 20 seeds. The purple and green colors refer to the two discrete actions (move left or right). Our ProFQI approximates the optimal policy more closely.

Projective Fitted $Q$-Iteration

We initially evaluate ProFQI on the car-on-hill problem. As done in Ernst, Geurts, and Wehenkel (2005), we measure performance by generating roll-outs starting from different states on a grid of size $17 \times 17$, and accounting for the fact that the dataset $D$ 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). First, the benefit of PBO is observable in the quality of the policy computed by FQI and ProFQI (Figure 8). 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 8a (Ernst, Geurts, and Wehenkel 2005)), while FQI is significantly more inaccurate. The consequences of this are reflected in the performance shown in Figure 9, that are obtained with FQI and ProFQI for three different values of Bellman iterations $K$ (black 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 9 are the regular iterations for FQI and the applications of the trained PBO for ProFQI. Thus,

Experiments

We consider both ProFQI and ProDQN, comparing their performance with their regular counterparts\(^4\). To handle the increased complexity of the input space of the considered problems, we leverage neural network regression to model our PBO. We consider an offline setting, where we use ProFQI on car-on-hill (Ernst, Geurts, and Wehenkel 2005), and an online setting, where we use ProDQN on bicycle balancing (Randlov and Alstrøm 1998), and lunar lander (Brockman et al. 2016). We want to answer the following research question: does PBO enable moving toward the fixed point more effectively than the empirical Bellman operator?

\(^4\)The code is available at https://github.com/theovincent/PBO
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 ProFQI 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 FQI. It is interesting to note that the performance of ProFQI remains stable for subsequent applications of PBO after the number of iterations used for training.

Projected Deep Q-Network

We also evaluate PBO in an online setting, using our ProDQN algorithm and comparing against DQN (Mnih et al. 2015). We consider a bicycle balancing (Randlov and Alstrøm 1998) problem and the lunar lander environment (Brockman et al. 2016). We set the number of Bellman iterations $K = 8$ for bicycle balancing and $K = 10$ for the 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 possibilities for effective exploration strategies based on PBO, that we postpone to future works.

Figure 10 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. Figure 10a shows that contrary to car-on-hill, on the bicycle balancing task ProDQN improves slower than DQN, but it keeps increasing performance after $K$ iterations thanks to PBO. This behavior illustrates that PBO can be used after the training to move the parameters in a favorable direction (see Figure 2a (blue trajectory)). Similarly, Figure 10b shows that, on lunar lander, ProDQN keeps increasing ever so slightly, but it is also stronger than DQN overall.

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 called the projected Bellman operator can generate a sequence of parameters that can progressively approach the ones of the optimal action-value function. We formulated an optimization problem and an algorithmic implementation to learn PBO in offline and online RL. 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). Nevertheless, recent advances in deep learning methods for learning operators (Kovachki et al. 2021, 2023) can provide a promising future direction.
Acknowledgments

This work was funded by the German Federal Ministry of Education and Research (BMBF) (Project: 01IS22078). This work was also funded by Hessian.ai through the project ‘The Third Wave of Artificial Intelligence – 3AI’ by the Ministry for Science and Arts of the state of Hessen and by the grant “Einrichtung eines Labors des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an der Technischen Universität Darmstadt”. This paper was also supported by FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, Investment 1.3, Line on Artificial Intelligence).

References


