Low-rank State-action Value-function Approximation

Sergio Rozada, Victor Tenorio, and Antonio G. Marques*

***Abstract*—Value functions are central to Dynamic Programming and Reinforcement Learning but their exact estimation suffers from the curse of dimensionality, challenging the development of practical value-function (VF) estimation algorithms. Several approaches have been proposed to overcome this issue, from non-parametric schemes that aggregate states or actions to parametric approximations of state and action VFs via, e.g., linear estimators or deep neural networks. Relevantly, several high-dimensional state problems can be well-approximated by an intrinsic low-rank structure. Motivated by this and leveraging results from low-rank optimization, this paper proposes different stochastic algorithms to estimate a low-rank factorization of the $Q(s,a)$ matrix. This is a non-parametric alternative to VF approximation that dramatically reduces the computational and sample complexities relative to classical Q-learning methods that estimate $Q(s,a)$ separately for each state-action pair.**

***Index Terms*—Reinforcement Learning, Value Iteration, Q-Learning, Low-Rank Approximation, Stochastic Dynamic Programming.**

I. Introduction and motivation

As complex interconnected systems and big data become pervasive, the engineering goal is to design intelligent algorithms that leverage the data and learn how to interact with the world autonomously. Reinforcement Learning (RL) tries to embed into algorithms how humans interact with the world, learning in real-time by trial and error [1]–[3]. Technically speaking, RL solves sequential optimization problems, like Dynamic Programming (DP) [2], in a model-free fashion. When the environment is "simple", solid theoretical results exist, and a range of algorithms address how to learn the mapping from the state observed at every time instant to the action to be executed. However, RL is algorithmically challenging when the actions and states describing the environment are numerous, high-dimensional, or defined in continuous domains. Such a curse of dimensionality calls for approximated algorithms that, e.g., discretize continuous domains while simultaneously limiting the complexity of the problem to be solved [1]. Indeed, different (parametric and nonparametric) approaches to reduce the number of degrees of freedom (keeping computational complexity under control and facilitating learning) at the cost of sacrificing optimality exist [2]. Motivated by its practical relevance and the described computational challenges, this paper puts forth nonparametric, stochastic, model-free algorithms that leverage results from low-rank optimization and matrix completion [4], [5]. The rest of this section provides details on the notation and fundamentals of RL and, once those have been presented, describes the contribution in a more rigorous manner.

This work was supported by the Spanish Federal Grants SPGRAPH (PID2019-105032GB) and by the Young Researchers R&D Project Grants F661-MAPPING-UCI and F663-AAGNCS both funded by the Community of Madrid (CAM) and the King Juan Carlos University (URJC). All the authors are with the Dept. of Signal Theory and Comms., URJC, Madrid, Spain. Sergio Rozada is also with Clients Solutions Advanced Analytics, BBVA, Madrid, Spain. *Contact author: antonio.garcia.marques (AT) urjc.es.

Fundamentals of RL and notation. We deal with closed-loop setups where agents interact sequentially with an environment. The three key elements to define the RL problem are the states describing the environment, the actions agent(s) can take, and their associated rewards. Mathematically, the environment is represented by a (discretized) set of states $\mathcal{S}$. In each state, the agent takes actions from a (discretized) set of actions, or action space, $\mathcal{A}$. We assume a time-slotted scenario and, with $t = 1, ..., T$ representing the time index. After taking an action a_t in a particular state s_t, the agent receives a numerical signal, or reward r_t, that quantifies the instantaneous value of that state-action pair. Two key aspects in RL are: (i) the dependence of r_t on s_t and a_t is oftentimes not deterministic, and (ii) the action a_t has an impact not only in r_t but also in $s_{t'}$ for $t' > t$, coupling the optimization across time. In other words, when deciding the value of a_t one must take into account not only its impact on r_t but also on $r_{t'}$ for $t > t'$. A common approach to deal with these two issues is to rely on Markovianity, recast first the optimization within the framework of Markov Decision Process (MDP) and, then, view RL as an stochastic approach for the MDP.

In this context, suppose that we have a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ that maps states into actions and define the expected aggregated reward as $\mathbb{E}[\sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau | s_t, a_t]$, where $\gamma \in (0,1)$ is a discount factor that places more focus on near-future values, and the time-horizon is oftentimes set to $T = \infty$ [1], [2]. Given the policy π, the so-called value function (VF) quantifies the expected reward associated with that particular (alternatively, the optimal) policy. Since the expectation depends on both the state and the action, two versions of the VF can be defined. For the one used in this paper, the so-called state-action value function (SA-VF) $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the dependence on the state and the action is kept. Since each (s,a) pair has one Q-value associated with it, when the states and actions are discrete, all the Q-values are arranged in the form of a matrix $\mathbf{Q}^\pi \in \mathbb{R}^{D_\mathcal{S} \times D_\mathcal{A}}$, with D_S being the number of states (the cardinality of $\mathcal{S}$) and D_A the cardinality of $\mathcal{A}$ [1]. When the state (action) is d_S-dimensional and continuous, the classical approach is to discretize each dimension using N_S intervals so that $D_S = N_S^{d_S}$. This exponential dependence affects computational and learning performance, calling for algorithms to keep the complexity of the model under control.

The distinctive feature in RL is that the statistical dependence across states and actions is unknown (or intractable), and one must resort to model-free algorithms that learn on-the-fly by using realizations (trajectories) of the MDP [1], [2]. As in MDP and DP, there are RL methods focused on learning the policy π –typically under a parametric approach followed by stochastic gradient descent (SGD)–, while others focus on learning the VF (using parametric or nonparametric approaches). Stochastic optimization of neural networks (NN) and linear-based models are widely-used parametric alterna-

tives [1], [2], while the classical Q-learning algorithm is the most celebrated example of the non-parametric class [6].

Contribution and related work. This manuscript proposes different *low-rank stochastic* RL algorithms to estimate the Q-function in a *model-free* and *online* setup. Most of the proposed algorithms are designed using SGD and can be interpreted as temporal differences (TD)-based schemes [2]. Once the estimation concludes, the policies are obtained as those that, for the current state, select the action maximizing the estimated SA-VF. The key aspect of the design is to regularize the estimation problem by promoting an SA-VF with a low-rank structure via matrix factorization. Those techniques have been successfully utilized in the context of low-rank optimization and matrix completion [4], [5], [7], [8], but their use in DP/MDP (and, especially, in RL and TD) has been limited. For example, [9], [10] used matrix-factorization approaches to approximate the transition matrix of an MDP and, then, leverage those to obtain the associated VF and optimal policy. In [11], the author proposes obtaining first the Q-function of an MDP and, then, approximate it using a low-rank plus sparse decomposition. Similarly, in the context of DPs associated with energy storage, [12], [13] proposed a low-rank (rank-one) approximation for the estimation of the state VF. Recently, low-rank optimization has been proposed to approximate the SA-VF using *model-based* and *offline* setups [14], [15]. Another work that uses factorization techniques in the context of RL is [16], where the focus is on deep factorized architectures amenable to be implemented distributedly in multi-agent setups. Also within RL setups, *linear models* have been used to approximate the SA-VF on-the-fly [17]. In those works a set of features is defined (based on prior knowledge, collected data, or spectral properties of the transition matrices [18], [19]) and then each entry of the SA-VF is modeled as a weighted sum of the features associated with the (s,a) pair. The weights are assumed to be the same across Q and the focus of the RL algorithm is on their estimation using sample trajectories. As discussed in detail at the end of Sec. 2, a distinctive feature of the low-rank RL approaches we put forth in this paper is that, here, the features are assumed to follow a factorized (bilinear) model and both features and weights are learned jointly, in real-time, from the observed data.

II. Approximate low-rank state-action value function estimation

Before presenting our algorithms, we need to provide a succinct description of classical nonparametric RL estimation of the Q-function, which tries to optimize:

$$\hat{\mathbf{Q}} = \text{argmin}_{\mathbf{Q}} \, \tfrac{1}{2} \textstyle\sum_{(s,a)\in\mathcal{M}} (q_s^a - [\mathbf{Q}]_{s,a})^2 \tag{1}$$

where $\mathcal{M}$ is the set of state-action tuples (s,a) that the agent has sampled from the environment, and q_s^a is the target signal when being in state s and taking action a. In contrast to other learning paradigms, in RL this target signal q_s^a is not known in advance and should be estimated on the fly. To estimate the target signal, we can use the available reward to define $q_s^a = r_s^a + \gamma[\hat{\mathbf{Q}}]_{s',a'}$, where the tuple (s',a') represents the state-action pair sampled in the subsequent time instant.

Suppose now that: i) the action a' is selected as the one maximizing the current estimate of the SA-VF and, hence, $[\hat{\mathbf{Q}}]_{s',a'} = \max_a [\hat{\mathbf{Q}}]_{s',a}$; and ii) (1) is solved via an SGD method that, at each time t, updates the current estimates based on (s_t, a_t, r_t, s_{t+1}). This yields the following update rule

$$\begin{aligned}[\hat{\mathbf{Q}}^t]_{s_t,a_t} =& [\hat{\mathbf{Q}}^{t-1}]_{s_t,a_t} + \alpha_t \big(r_t + \gamma \max_a [\hat{\mathbf{Q}}^{t-1}]_{s_{t+1},a} \\ & - [\hat{\mathbf{Q}}^{t-1}]_{s_t,a_t}\big)\end{aligned} \tag{2}$$

with α_t being the learning rate (stepsize), $\hat{\mathbf{Q}}^t$ being the estimate of the SA-VF at time t, and where we remark that $[\hat{\mathbf{Q}}^t]_{s,a} = [\hat{\mathbf{Q}}^{t-1}]_{s,a}$ for all $(s,a) \neq (s_t,a_t)$. The scheme in (2) corresponds to the celebrated (TD-based) Q-learning algorithm, which enjoys convergence guarantees, provided that all (s,a) pairs are visited infinitely often [6]. To facilitate this latter point, an exploration module is added to the algorithm so that with probability $(1-\varepsilon_t)$ the action $a' = \text{argmax}_a [\hat{\mathbf{Q}}^{t-1}]_{s_{t+1},a}$ is selected (and, hence, the expression in (2) applies) and, with probability ε_t, the action a' is chosen uniformly at random. Unfortunately, Q-learning can take a long time to converge, the reason being that the number of parameters $D_S D_A$ can be very large and that, when a pair (s,a) is observed at time t, only one of the entries of the matrix $\hat{\mathbf{Q}}^t \in \mathbb{R}^{D_S \times D_A}$ is updated.

Low-rank via matrix factorization. Our contribution to facilitate and accelerate the learning of the SA-VF is to limit its degrees of freedom by promoting low-rank solutions. More specifically, we modify the optimization (estimation) of $\mathbf{Q}$ to either force or promote low-rank estimates. Since adding the constraint $\text{rank}(\mathbf{Q}) \leq M$ to the optimization in (1) –or, alternatively, adding $\text{rank}(\mathbf{Q})$ as a regularizer– is computationally intractable, one can replace $\text{rank}(\mathbf{Q})$ with the nuclear norm $\|\mathbf{Q}\|_*$, its convex surrogate, or implement a non-convex matrix factorization approach [5]. The latter entails considering the matrices $\mathbf{L} \in \mathbb{R}^{D_S \times M}$ and $\mathbf{R} \in \mathbb{R}^{M \times D_A}$, write the VF as $\mathbf{Q} = \mathbf{L}\mathbf{R}$, and solving

$$\{\hat{\mathbf{L}}, \hat{\mathbf{R}}\} = \text{argmin}_{\mathbf{L},\mathbf{R}} \, \tfrac{1}{2} \textstyle\sum_{(s,a)\in\mathcal{M}} (q_s^a - [\mathbf{L}\mathbf{R}]_{s,a})^2. \tag{3}$$

The approximated SA-VF is then obtained as $\hat{\mathbf{Q}} = \hat{\mathbf{L}}\hat{\mathbf{R}}$, which is guaranteed to have a rank no larger than M. While the factorized problem in (3) is non-convex, computationally efficient alternating-minimization schemes that, in the context of matrix completion, are guaranteed to converge to a local minimum can be implemented [5], [7]. While different stochastic algorithms can be developed to handle (3), since the cost is quadratic and the problem is bilinear in $\mathbf{L}$ and $\mathbf{R}$, a stochastic alternating least-squares approach is well motivated. Let $\hat{\mathbf{L}}^t$ and $\hat{\mathbf{R}}^t$ denote the stochastic estimates of $\mathbf{L}$ and $\mathbf{R}$ at the end of the slot t; define the matrix $\mathbf{Q}^t \in \mathbb{R}^{D_S \times D_A}$ as $[\mathbf{Q}^t]_{s,a} = [\hat{\mathbf{L}}^{t-1}(\hat{\mathbf{R}}^{t-1})]_{s,a}$ for all $(s,a) \neq (s_t,a_t)$ and $[\mathbf{Q}^t]_{s_t,a_t} = q_s^a = r_t + \gamma \max [\hat{\mathbf{L}}^{t-1}(\hat{\mathbf{R}}^{t-1})]_{s',a}$; and let $k = 1, ..., K$ be an iteration index. With this notation at hand, initialize $\hat{\mathbf{R}}_{[0]} = \hat{\mathbf{R}}^{t-1}$, set $\bar{\mathbf{Q}} = \mathbf{Q}^t$, and iterate as

$$\hat{\mathbf{L}}_{[k]} = \bar{\mathbf{Q}} \hat{\mathbf{R}}_{[k-1]}^{\mathsf{T}} (\hat{\mathbf{R}}_{[k-1]} \hat{\mathbf{R}}_{[k-1]}^{\mathsf{T}})^{-1} \text{ and} \tag{4}$$

$$\hat{\mathbf{R}}_{[k]} = (\hat{\mathbf{L}}_{[k]}^{\mathsf{T}} \hat{\mathbf{L}}_{[k]})^{-1} \hat{\mathbf{L}}_{[k]}^{\mathsf{T}} \bar{\mathbf{Q}} \tag{5}$$

for $k = 1, ..., K$ and, finally, set the stochastic estimates to $\hat{\mathbf{L}}^t = \hat{\mathbf{L}}_{[K]}$ and $\hat{\mathbf{R}}^t = \hat{\mathbf{R}}_{[K]}$. A simpler alternative to optimize (3) is to run just a single iteration of a stochastic gradient descent. To that end, let $[\hat{\mathbf{L}}^t]_{s_t} \in \mathbb{R}^M$ be a vector collecting the M entries of $\hat{\mathbf{L}}^t$ associated with state s_t and, likewise, let $[\hat{\mathbf{R}}^t]_{a_t} \in \mathbb{R}^M$ be a vector collecting the M entries of $\hat{\mathbf{R}}^t$ associated with action a_t. With these conventions, at time t, we use (s_t, a_t, s_{t+1}) to update the stochastic estimates as follows

$$[\hat{\mathbf{L}}^t]_{s_t} = [\hat{\mathbf{L}}^{t-1}]_{s_t} + \alpha_t \big(r_t + \gamma \max_a [\hat{\mathbf{L}}^{t-1}\hat{\mathbf{R}}^{t-1}]_{s_{t+1},a} - [\hat{\mathbf{L}}^{t-1}\hat{\mathbf{R}}^{t-1}]_{s_t,a_t}\big)[\hat{\mathbf{R}}^t]_{a_t} \quad (6)$$

$$[\hat{\mathbf{R}}^t]_{a_t} = [\hat{\mathbf{R}}^{t-1}]_{a_t} + \alpha_t \big(r_t + \gamma \max_a [\hat{\mathbf{L}}^{t-1}\hat{\mathbf{R}}^{t-1}]_{s_{t+1},a} - [\hat{\mathbf{L}}^{t-1}\hat{\mathbf{R}}^{t-1}]_{s_t,a_t}\big)[\hat{\mathbf{L}}^t]_{s_t}, \quad (7)$$

and $[\hat{\mathbf{L}}^t]_s = [\hat{\mathbf{L}}^{t-1}]_s$ and $[\hat{\mathbf{R}}^t]_a = [\hat{\mathbf{R}}^{t-1}]_a$ for all $(s, a) \neq (s_t, a_t)$. The updates in (6)-(7) resemble those of the Q-learning algorithm in (2), are less computationally demanding than those in (4), and for sufficiently small α_t converge to a local optimum. In contrast, the convergence of the alternating scheme (which is not always guaranteed) happens at a faster pace. If the speed of convergence is an issue, the stepsize in (6)-(7) can incorporate a gradient normalization to circumvent the slower convergence due to plateaus and saddle points oftentimes present in non-convex optimization [20]. Finally, all the methods proposed here consider an exploration module to guarantee that all (s, a) pairs are visited.

Compared to Q-learning, the algorithms just described exhibit two main advantages: a) the number of parameters to estimate is smaller –$(D_S + D_A)M$ vs. $D_S D_A$– and b) at each t, the entire row $[\hat{\mathbf{L}}^t]_{s_t}$ and column $[\hat{\mathbf{R}}^t]_{a_t}$ (with M values each) are updated. This contrasts with Q-learning, where only a single parameter, the entry $[\hat{\mathbf{Q}}^t]_{s_t,a_t}$, is updated at time t.

Matrix factorization meets the nuclear norm. Leveraging the fact that the nuclear norm can be written as $\|\mathbf{Q}\|_* = \frac{1}{2} \min_{\mathbf{LR}=\mathbf{Q}} \|\mathbf{L}\|_F^2 + \|\mathbf{R}\|_F^2$, the cost in (3) can be augmented with the Frobenius regularizers $\eta\|\mathbf{L}\|_F^2$ and $\eta\|\mathbf{R}\|_F^2$. The regularizers are quadratic; hence, the structure and computational complexity of the updated stochastic iterates is roughly the same. On the other hand, since the nuclear norm is convex, the regularizers help to stabilize the behavior of our stochastic algorithms. In fact, conditions under which the Frobenius-regularized iterates converge to the same solution than the nuclear-norm minimization can be rigorously obtained [8], [21]. Lastly, the regularized formulation promotes solutions whose rank is typically smaller than M. This is important because it bypasses the problem of selecting the value of M.

Comparison with parametric linear estimation of the SA-VF. Linear parametric schemes for SA-VF estimation consider the approximation $[\hat{\mathbf{Q}}]_{s,a} = Q_\theta(s, a) = \boldsymbol{\phi}(s, a)^T \boldsymbol{\theta}$, where $\boldsymbol{\phi}(s, a) \in \mathbb{R}^M$ is the set of M features that defines a particular state-action pair and $\boldsymbol{\theta} \in \mathbb{R}^M$ are the parameters/weights to be estimated online [18]. The key issue in these approaches is how to pre-design the feature vectors $\boldsymbol{\phi}(s, a)$. Differently, we learn the feature vectors and the weights jointly. To be specific, suppose that $\text{rank}(\hat{\mathbf{Q}}) = M$ and let $\mathbf{v}_m$ ($\mathbf{u}_m$) be the mth left (right) singular vector and σ_m the mth singular value. Then, we have that $[\hat{\mathbf{Q}}]_{s,a} = \sum_{m=1}^M \sigma_m [\mathbf{v}_m]_s [\mathbf{u}_m]_a$, illustrating that in our approach the features are implicitly defined as $\boldsymbol{\phi}(s, a) = [[\mathbf{v}_1]_s [\mathbf{u}_1]_a, ..., [\mathbf{v}_M]_s [\mathbf{u}_M]_a]^T$ and the linear weights as $\boldsymbol{\theta} = [\sigma_1, ..., \sigma_M]^T$, with the difference being that here both $\boldsymbol{\phi}(s, a)$ and $\boldsymbol{\theta}$ are learned online.

Reshaping the Q-matrix. In many applications, the states and, oftentimes, the actions involve multiple variables. To be mathematically precise, suppose that we have d_S state variables (dimensions) and d_A action variables. If the ith state variable takes values from the set $\mathcal{S}_i$, the state space is defined as $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_{d_S}$. Analogously, we have that $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_{d_A}$. Even in those cases, the RL literature refers to $Q(s, a)$ as a two-dimensional function, arranging its values in the form of a matrix $\mathbf{Q}$ whose rows represent then tuples of states and its columns tuples of actions. Throughout this paper, we kept such a convention and developed algorithms to learn a low-rank approximation of $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$, which, because since $|\mathcal{A}|$ is typically much smaller than $|\mathcal{S}|$, is a (sometimes extremely) tall matrix.

We propose here an alternative low-rank approximation scheme that first reshapes $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ as $\bar{\mathbf{Q}} \in \mathbb{R}^{N_R \times N_C}$, with $N_R N_C = |\mathcal{S}||\mathcal{A}|$, and then implements a low-rank decomposition on the reshaped matrix $\bar{\mathbf{Q}}$. The idea in the first step is to have $N_R \approx N_C \approx \sqrt{|\mathcal{S}||\mathcal{A}|}$, so that the reshaped matrix $\bar{\mathbf{Q}}$ is approximately square, reducing the number of parameters to be estimated in the low-rank approximation run in the second step from $P_{\mathbf{Q}} = M(|\mathcal{S}| + |\mathcal{A}|)$ to $P_{\bar{\mathbf{Q}}} = M(N_C + N_R) \approx 2M\sqrt{|\mathcal{S}||\mathcal{A}|}$. For the practical setups where $|\mathcal{S}| \gg |\mathcal{A}|$ we can split the d_S state variables into two sets and consider a $\bar{\mathbf{Q}}$ matrix whose rows correspond to an element of $\mathcal{S}_1 \times ... \times \mathcal{S}_{d_{S'}}$, whose columns correspond to an element of $\mathcal{S}_{d_{S'}+1} \times ... \times \mathcal{S}_{d_S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_{d_A}$, and whose number of columns and rows is approximately the same. Unless prior knowledge exists, the ordering of the state and action variables is arbitrary and, hence, the assignment of state variable to columns and rows is arbitrary as well, potentially leading to different approximation performances.

While conceptually simple, the numerical results will demonstrate the benefits and robustness of the proposed approach in standard RL test-cases. Arranging the values of the Q-matrix in the form of a tensor with $d_S + d_A$ dimensions and postulating low-rank tensor decomposition algorithms emerge as a natural follow-up research direction.

III. Numerical experiments

This section tests the proposed algorithms in three of the standard RL problems of the toolkit OpenAI Gym (see [22] and Fig. 1.a). Our goal is to illustrate i) the convergence properties of our schemes; ii) the advantages of considering a low-rank $Q(s, a)$ matrix; and iii) the success of our method in an environment too large for classical non-parametric methods. The code and full details can be found in [23].

Convergence properties. We test the rate of convergence of (3) in the FrozenLake-v0 environment [22]. This is a simple deterministic finite-state finite-action grid-like environment, where a reward is given to the agent for reaching a goal state. Although it can be solved using Q-learning, our algorithm can exploit the low-rank structure of the environment

to accelerate the rate of convergence using almost half of the parameters. Convergence is analyzed via i) the number of episodes required to obtain the optimal solution; and ii) measuring the evolution of Squared Frobenius Error (SFE) between the estimated $\hat{\mathbf{Q}}$ and the true $\mathbf{Q}$ as $\text{SFE} = ||\hat{\mathbf{Q}} - \mathbf{Q}||_F^2$.

We run 100 simulations for different values of $\epsilon_t = \epsilon$. Q-learning stores a 16×4 state-action table, so that the total number of parameters is 64. When testing our low-rank alternative we impose a maximum rank of $M = 2$, so that the total number of parameters is $(16+4)M = 40$. The two main observations that can be obtained from Fig. 1.b and 1.c are that the low-rank alternatives: 1) can converge to the optimal solution, both in terms of the number of steps and SFE; and 2) converge faster in all possible scenarios, that is, for a fixed value of ϵ, the low-rank approach proposed in (3) requires fewer episodes than Q-learning to converge.

Parametrization efficiency. In the Pendulum-v0 environment [22], an agent tries to maintain a pendulum upright. This problem has two continuous states (angle and angular velocity), one continuous action, and the reward is multi-objective (keeping the pendulum vertical while minimizing the action exerted). Tackling the continuous-defined problem requires discretizing the state-action space. However, the finer the discretization, the larger the size of the $Q(s, a)$-matrix. In particular, the Cartesian product of the regularly-sampled states defines a discrete state space of size $D_S = 2121$ (see [23] for details). Five RL schemes have been compared with different discretizations of the action space. Two of them are Q-learning with the action space discretized at low and high resolutions, with size $D_A = 5$ and $D_A = 41$ respectively. Two are low-rank schemes that use high-resolution discretization ($D_A = 41$ actions), keep the shape of the Q matrix, and consider a (low) rank of $M = 3$ and $M = 5$, respectively. The last scheme uses the same resolution, but reshapes the Q matrix (with 295 rows and 295 columns) and sets the rank to $M = 10$. Note that imposing low rank brings down the number of parameters significantly. A roughly-discretized version of Q-learning uses $10,605$ parameters while the finer version needs $86,961$ parameters. The low-rank alternative with rank $M = 3$ only needs $6,486$ parameters while the larger version with $M = 5$ needs $10,810$ parameters. Finally, the reshaped alternative, which is closer to a square matrix, requires $5,900$ parameters for $M = 10$. Variants of Q-learning undersampling the state space did not converge. Notice also that the roughly-discretized version of the Q-learning with $D_A = 5$ actions and our low-rank scheme with $M = 5$ entail basically the same number of parameters. The low-rank scheme with rank $M = 3$ and the reshaped one with $M = 10$ were trained with the stochastic algorithm defined in (6) and (7) while the low-rank scheme with the larger rank $M = 5$ was trained with the regularized alternative. A certain level of exploration is required, thus the exploratory probability was set to $\epsilon_t = 0.2$.

Each algorithm was trained 10 times, and the results obtained can be found in Figs. 1.d and 1.e. These results are obtained running a greedy test episode every several training episodes. As expected, the smaller version of Q-learning converges faster to a solution able to keep the pendulum upright, but fails at selecting low-cost actions (the poor discretization of the action space forces the agent to select sub-optimal actions). In contrast, the proposed low-rank schemes converge faster than the Q-learning version with the largest number of parameters. For the simulated number of episodes, they also outperform both versions of Q-learning in terms of cost. Moreover, an SVD decomposition of the Q-matrix estimated by the Q-learning algorithm reveals that the first three singular values account for more than 80 % of the total variance. On the other hand, the reshaped scheme is the best performing agent. A more prudently designed parametrization of the model leads to better results in terms of both, cumulative rewards and speed of convergence. These results demonstrate that the (unconstrained) optimal solutions are inherently low-rank and, hence, our algorithms fruitfully exploit this structure.

Comparison against the state-of-the-art. The Acrobot-v1 is a double-pendulum environment that mimics a gymnast trying to swing up [22]. There are six state variables: the sine and cosine of the angle and the angular velocity of the upper and lower joints. Only one action variable exists, which can take three different values. The number of discrete states produced by the Cartesian product of regularly-sampled states grows exponentially on the number of states. Even discretizing each of the six states at low-resolution results in roughly $7 \cdot 10^6$ states, leading to a $Q(s, a)$-matrix with more than $20 \cdot 10^6$ entries (see [23] for details). This dimensionality explosion can be addressed using an intelligent reshaping of the $Q(s, a)$ into a 4520×4519 matrix and applying the LR scheme imposing low rank $M = 2$. The total number of parameters drops dramatically to $18,078$. Two versions of the LR algorithm are compared in this experiment, one that normalizes the stochastic gradient update and another one that does not.

The VF and SA-VF-based approaches to handle high-dimensional environments are limited to parametric estimators. Here we will compare our performance with that of Deep Q-learning (DQN) [3], an offline NN state-of-the-art representative of parametric value-based methods. Linear methods were also tested but their performance is not shown because their value was worse and required many more episodes (more than one additional order of magnitude) to converge. NN-based RL algorithms suffer from convergence problems due to the temporal correlation. This is typically overcome by using experience replay (ER) [3], which requires the training to be offline. A one-layer Multi-Layer Perceptron (MLP) with a comparable number of parameters is used to implement DQN. One simulation implements a more efficient version of DQN using mini-batches of size $S = 12$ obtained from the ER buffer to perform each training update. A lighter version of DQN is trained using mini-batches of size $S = 1$.

The results of 10 trials are shown in Fig. 1.f. As expected, the *parametric* NN-based schemes converge to a better solution. However, our normalized *non-parametric* LR alternative successfully handles the problem and, in fact, converges faster than the light version of DQN. In contrast to DQN, the LR algorithm is fully online and does not need ancillary offline twists to stabilize the training. It is also important to note that the DQN trained with mini-batches of size $S = 12$ took 2.5 more training time: the use of larger batches introduces a trade-off between the training time and efficient use of previous

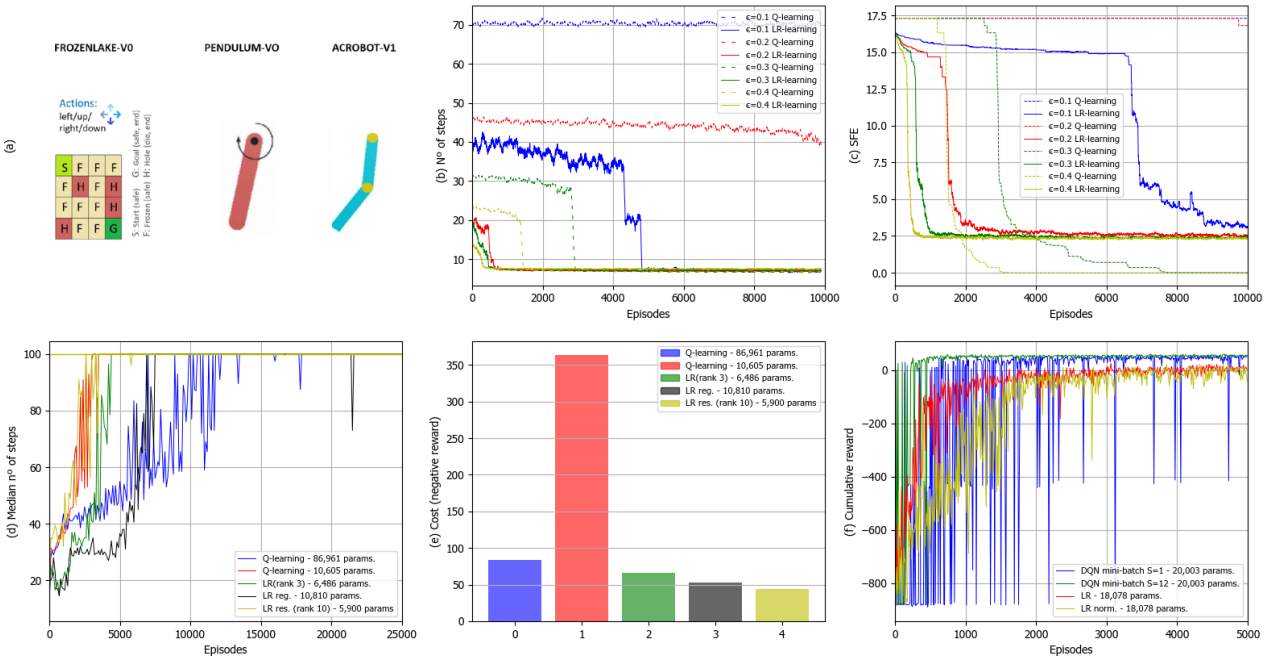

Fig. 1: The upper left picture (a) exemplifies the three tested RL environments, from left to right: FrozenLake-v0, Pendulum-v0 and Acrobot-v1. The two upper right pictures (b, c) show the performance of Q-learning and our low-rank (LR) alternative in FrozenLake-v0 for different values of ϵ. The two bottom left pictures (d, e) show the performance of Q-learning and the LR alternative in Pendulum-v0. The bottom right picture (f) depicts the performance of DQN and the LR alternative in Acrobot-v1.

experience. The effect of the normalization of the gradient can be observed as well. Both LR variants converge to the same solution, but the normalized one does it faster.

IV. Concluding Remarks

A collection of RL algorithms that exploit the intrinsic low-rank structure of the SA-VF has been proposed. The schemes leverage sounded matrix completion results to estimate the Q-function in a *model-free* and *online* fashion. Moreover, a reshaping of the Q-matrix has been introduced to parametrize the model in a more efficient way. The numerical experiments show the advantages of the low-rank models both in terms of speed of convergence and the achieved cumulative reward.

References

[1] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[2] D. P. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[5] I. Markovsky, *Low rank approximation*. Springer, 2019.

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[7] M. Udell, C. Horn, R. Zadeh, S. Boyd *et al.*, "Generalized low rank models," *Foundations and Trends® in Machine Learning*, vol. 9, no. 1, pp. 1–118, 2016.

[8] M. Mardani, G. Mateos, and G. B. Giannakis, "Decentralized sparsity-regularized rank minimization: Algorithms and applications," *IEEE Trans. Signal Processing*, vol. 61, no. 21, pp. 5374–5388, 2013.

[9] A. M. Barreto, R. L. Beirigo, J. Pineau, and D. Precup, "Incremental stochastic factorization for online reinforcement learning," in *Proc. AAAI Conf. Artificial Intelligence*, 2016.

[10] N. Jiang, A. Krishnamurthy, A. Agarwal, J. Langford, and R. E. Schapire, "Contextual decision processes with low bellman rank are pac-learnable," in *Proc. Intl. Conf. Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1704–1713.

[11] H. Y. Ong, "Value function approximation via low-rank models," *arXiv preprint arXiv:1509.00061*, 2015.

[12] B. Cheng and W. B. Powell, "Co-optimizing battery storage for the frequency regulation and energy arbitrage using multi-scale dynamic programming," *IEEE Trans. Smart Grid*, vol. 9.3, pp. 1997–2005, 2016.

[13] B. Cheng, T. Asamov, and W. B. Powell, "Low-rank value function approximation for co-optimization of battery storage," *IEEE Trans. Smart Grid*, vol. 9.6, pp. 6590–6598, 2017.

[14] Y. Yang, G. Zhang, Z. Xu, and D. Katabi, "Harnessing structures for value-based planning and reinforcement learning," *arXiv preprint arXiv:1909.12255*, 2019.

[15] D. Shah, D. Song, Z. Xu, and Y. Yang, "Sample efficient reinforcement learning via low-rank matrix estimation," *arXiv preprint arXiv:2006.06135*, 2020.

[16] Y. Chen, M. Zhou, Y. Wen, Y. Yang, Y. Su, W. Zhang, D. Zhang, J. Wang, and H. Liu, "Factorized Q-learning for large-scale multi-agent systems," *arXiv preprint arXiv:1809.03738*, 2018.

[17] F. S. Melo and M. I. Ribeiro, "Q-learning with linear function approximation," in *Intl. Conf. Comp. Learning Theory*. Springer, 2007, pp. 308–322.

[18] B. Behzadian, S. Gharatappeh, and M. Petrik, "Fast feature selection for linear value function approximation," in *Proc. Intl. Conf. Automated Planning and Scheduling*, vol. 29, no. 1, 2019, pp. 601–609.

[19] B. Behzadian and M. Petrik, "Low-rank feature selection for reinforcement learning." in *ISAIM*, 2018.

[20] R. Murray, B. Swenson, and S. Kar, "Revisiting normalized gradient descent: Fast evasion of saddle points," *IEEE Trans. Automatic Control*, vol. 64, no. 11, pp. 4818–4824, 2019.

[21] S. Burer and R. D. Monteiro, "Local minima and convergence in low-rank semidefinite programming," *Mathematical Programming*, vol. 103, no. 3, pp. 427–444, 2005.

[22] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[23] V. T. S. Rozada and A. G. Marques, "Online code repository: Low-rank state-action value-function approximation," *https://github.com/sergiorozada12/low-rank-rl*.