

MATRIX LOW-RANK APPROXIMATION FOR POLICY GRADIENT METHODS

Sergio Rozada, and Antonio G. Marques

Dept. of Signal Theory and Communications, King Juan Carlos University, Madrid, Spain

ABSTRACT

Estimating a policy that maps states to actions is a central problem in reinforcement learning. Traditionally, policies are inferred from the so called value functions (VFs), but exact VF computation suffers from the curse of dimensionality. Policy gradient (PG) methods bypass this by learning directly a parametric *stochastic* policy. Typically, the parameters of the policy are estimated using neural networks (NNs) tuned via stochastic gradient descent. However, finding adequate NN architectures can be challenging, and convergence issues are common as well. In this paper, we put forth *low-rank* matrix-based models to estimate efficiently the parameters of PG algorithms. We collect the parameters of the stochastic policy into a matrix, and then, we leverage matrix-completion techniques to promote (enforce) *low rank*. We demonstrate via numerical studies how *low-rank* matrix-based policy models reduce the computational and sample complexities relative to NN models, while achieving a similar aggregated reward.

Index Terms— Reinforcement Learning, Low-rank optimization, Policy gradients, Actor-critic methods.

1. INTRODUCTION

In the era of big data, complex dynamical systems call for intelligent algorithms that use observations to adapt the way they behave with the world. Reinforcement Learning (RL) addresses this by implementing *stochastic* algorithms that learn by trial and error how to interact with the environment [1, 2]. Technically speaking, RL models decision-making problems: given an environment defined by a set of states, the RL goal is to learn the best action to be made in each state. Then, RL is about estimating the reward associated with state-action pairs, which is referred to as value function (VF), and inferring a function that maps the states into actions, which is referred to as policy. Typically, RL algorithms estimate first the VF (strictly speaking, the expected accumulated reward associated with each state-action pair) and, then, implement a policy that maximizes the VF. However, value-based methods are algorithmically challenging, and suffer from the curse of dimensionality. To overcome these issues, policy-based methods were introduced. They bypass these problems by estimating the policy directly. To render the design of these methods tractable, policies are typically assumed stochastic and parametric, so that the problem boils down to estimating the parameters of some probability distribution.

The workhorse approach in policy approaches is to consider normal distributions and learn the parameters of those Gaussians using a neural network (NN) architecture that uses the states as inputs [3].

Work supported by the Spanish NSF Grants SPGraph (PID2019-105032GB-I00/AEI/10.13039/501100011033) and DATRASCOOP@SESM (TED2021-130347B-I00). All the authors are with the Dept. of Signal Theory and Comms., King Juan Carlos University, Madrid, Spain. Email contact author: antonio.garcia.marques@urjc.es.

NNs, and therefore, most policy-based methods tend to suffer from convergence issues, as they strongly depend on the architecture employed. As a result, postulating (finding) the proper NN architecture for the RL setup at hand is usually a problem in itself.

In the context of stochastic policy methods for RL, this paper takes an alternative path and, leveraging matrix completion results [4, 5], puts forth a low-rank algorithm to estimate stochastic policy models. The ultimate goal is to design an estimation scheme that i) is sufficiently generic to learn the policy and ii) mitigates some of the problems present in NN-based schemes. The rest of this section is devoted to explaining the state of the art and the contribution in more detail. Fundamentals of RL are discussed in Section 2, and our new low-rank scheme is introduced in Section 3. Numerical experiments showcasing some of the benefits of our approach are provided in Section 4.

Detailed contribution and related work. This paper investigates the design of *low-rank* schemes for (actor/critic) policy RL methods leveraging matrix completion techniques. The central aspects of such a design are i) the consideration of a policy method based on an actor and a critic (where the latter estimates a VF); ii) modeling the actor parameters and the critic VF as matrices (typically associated with a clusterization of the different dimensions of the input state); and iii) to regularize the estimation problem by enforcing a low-rank structure via matrix factorization. Although those techniques are well-known in the context of low-rank optimization and matrix completion [4, 5, 6, 7], they have not been exploited in the context of policy-based RL methods. While closely-related ways of modeling parsimony in RL have been investigated, e.g. sparsity [8, 9], the use of low-rank optimization in the general context of RL has been limited. Notable exceptions include efforts to approximate some structures of the Markov Decision Process (MDP), such as the transition matrix or the reward function, as low rank [10, 11, 12]. Matrix-completion techniques have also been used to compress the estimated Q -functions of an MDP [13]. In energy storage applications, low-rank (rank-one) methods have been proposed for the estimation of the VF [14, 15]. Along the same lines, *linear models* have been used to approximate VFs on-the-fly [16]. These methods operate by defining a set of features [17] and then modeling the VF as a weighted sum of the features associated with the state-action pair. More recently, in the context of value-based methods, schemes that estimate the VF promoting a low-rank matrix structure [18, 19, 20] and a low-rank tensor structure [21] have been proposed.

2. FUNDAMENTALS OF RL AND NOTATION.

RL models the world as a closed-loop setup where agent(s) sequentially interact with the environment. The environment consists of i) the space of states \mathcal{S} ; ii) the space of action agent(s) can take \mathcal{A} ; and iii) the reward associated with every state-action pair. In each (typically multidimensional) state, the agent takes an action and obtains a reward. To be more specific, let $t = 1, \dots, T$ be a time index.

Given a particular state s_t , the agent takes an action a_t , and obtains a reward r_t . The reward r_t quantifies the *instantaneous* value of the state-action pair. However, the action a_t impacts $s_{t'}$ for $t' > t$ and, as a result, affects future rewards $r_{t'}$ for $t' > t$. This illustrates that actions are coupled across time and that the aggregated (long-term) reward must be considered for decision-making. Furthermore, the dependence of r_t on s_t and a_t is usually stochastic, rendering the optimization more challenging. Markovianity is commonly used to mitigate these issues, so that the optimization can be recast as a MDP.

In this context, i) a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ is a function that maps states into actions; and ii) the VF associated with (s_t, a_t) is the expected cumulative reward $\mathbb{E}[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} | s_t, a_t]$, with $\gamma \in (0, 1)$ being a discount factor that places more focus on short-term rewards [1, 2]. For a specific policy π , the VF quantifies the expected reward associated with a particular state by taking the actions dictated by π . Value-based methods focus on estimating the VFs (using parametric or non-parametric approaches) of the MDP, to infer a policy later on from the VFs by greedily following the path of highest value [22, 23]. Policy-based methods, on the other hand, focus on directly learning the policy π , typically using a stochastic parametric models [24, 25]. To estimate the parameters, a set of trajectories sampled from the environment is assumed to be available, and those are used to run a stochastic gradient descent scheme. Each trajectory $\tau_T : (s_0, a_0, r_0), \dots, (s_T, a_T, r_T)$ is a set of state-action-rewards triplets obtained from the sequential interaction of an agent with the environment.

From an optimization perspective, policy-based methods aim at maximizing the reward of a trajectory $\mathcal{R}(\tau_T)$ over all possible trajectories τ . To be precise, consider the following objective function:

$$J(\theta) = \mathbb{E}_{\mathcal{T}_\theta(\tau_T)}[\mathcal{R}(\tau_T)] \quad (1)$$

where θ denotes the parameters of a policy π_θ . Then, $\mathcal{T}_\theta(\cdot)$ is the probability density function of the trajectories under the given policy π_θ . Leveraging Markovianity, it follows that $\mathcal{T}_\theta(\tau_T) = \mathcal{D}(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) P(s_{t+1} | s_t, a_t)$, where $\mathcal{D}(\cdot)$ is the initial distribution of the MDP. Since direct optimization of (1) is intractable, the standard approach is to estimate the parameters θ of the policy π_θ via (stochastic) gradient ascent methods [26]. The update rule takes the form $\theta_{h+1} = \theta_h + \alpha_\theta \nabla_\theta J(\theta_h)$, where h is the iteration index, and α_θ is the gradient step, or learning rate. Obtaining the gradient of J is challenging since it involves computing derivatives of an expectation across \mathcal{T}_θ . The policy gradient theorem [27] provides an elegant reformulation of $\nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{T}_\theta} \left[\sum_{t=0}^T \mathcal{R}(\tau_T) \nabla_\theta \log \pi_\theta(a_t | s_t) \right], \quad (2)$$

which is more tractable. One of the most celebrated policy-based algorithms is REINFORCE [24], or Monte-Carlo policy gradient, which samples a full trajectory from the environment to avoid computing the expectation, and uses (2) to update the parameters using the actual total return $G_t = \sum_{t'=t}^T r_{t'}$ in lieu of $\mathcal{R}(\tau_T)$.

Actor critic methods. The main drawback of Monte-Carlo methods is that they exhibit a high variance, due to so called credit assignment problem. This issue is usually addressed by subtracting a baseline value b_t from the return G_t to reduce the variance of the gradient estimate while keeping the bias unchanged [28]. A choice that provides several benefits is setting b_t to the VF. The quantity resultant from this subtraction is denoted as $A(s_t) = G_t - V(s_t)$ and is called advantage function. The price to pay in this case is

that a method to estimate the VFs is needed as well. This leads to a two-step method (actor-critic) that involves the estimation of the parameters of the policy π_θ (actor) as well as those of the VF (critic). The parameters of the actor θ are updated via gradient ascent using

$$\nabla_\theta J(\theta) = \sum_{t=0}^T A(s_t) \nabla_\theta \log \pi_\theta(a_t | s_t), \quad (3)$$

where $A(s_t) = G_t - V_\omega(s_t)$, and $V_\omega(s_t)$ is considered given. In contrast, the parameters ω of the VF are obtained as

$$\omega^* = \arg \min_{\omega} \mathcal{L}(\omega) := \arg \min_{\omega} \frac{1}{2} \sum_{t=0}^T (G_t - V_\omega(s_t))^2. \quad (4)$$

While some parametric models (4) can be solved in closed form, ω are typically estimated using gradient descent methods of the form

$$\omega_{h+1} = \omega_h - \alpha_\omega \nabla_\omega \mathcal{L}(\omega_h), \quad (5)$$

where h is the iteration index and α_ω is the gradient step.

Gaussian policies. In setups where the action state \mathcal{A} is continuous and one-dimensional, a commonly adopted approach is to model actions as samples of a univariate Gaussian distribution $a_t \sim \mathcal{N}(a | \mu(s_t), \sigma(s_t))$ [29, 30]. The problem of finding the optimal action associated with a state is reformulated as finding the mean $\mu(s_t)$ and the standard deviation $\sigma(s_t)$ associated with the state. The functions $\mu : \mathcal{S} \mapsto \mathbb{R}$ and $\sigma : \mathcal{S} \mapsto \mathbb{R}^+$ can be either parametric or non-parametric, with a stronger preference in the literature for the former. Clearly, to find the gradient on the right-hand side of (3) w.r.t. those parameters, one needs to resort to the chain rule and find first the derivative of J w.r.t. μ and σ . Upon setting the distribution of the policy to a Gaussian, this approach leads to the following partial derivatives

$$\nabla_\mu J(\mu, \sigma) = \sum_{t=0}^T A(s_t) \frac{a_t - \mu(s_t)}{\sigma(s_t)^2}, \quad (6)$$

$$\nabla_\sigma J(\mu, \sigma) = \sum_{t=0}^T A(s_t) \left(\frac{(a_t - \mu(s_t))^2}{2\sigma(s_t)^3} - \frac{1}{\sigma(s_t)} \right). \quad (7)$$

Practical implementations often consider simpler models for $\sigma(s_t)$. For example, it is not uncommon to assume that σ is constant and independent of the state s_t , reducing the problem to learn this single value (or even fixing it using prior knowledge).

After describing the basics of policy methods in RL, we are ready to introduce our scheme. The reader should notice that although in this paper we have focused on Gaussian policies, the proposed approach can be extended to other policy models.

3. POLICY GRADIENTS AND MATRIX FACTORIZATION

The state space \mathcal{S} in most MDPs is discrete, either because the number of states is finite or because it represents the discretized (sampled) version of a continuous space. As a result, we have a finite number N_S of states. Since each state is mapped to a mean and standard deviation, the goal in this section is to provide a (low-rank based) scheme that yields an efficient estimator for the N_S mean and standard deviation pairs.

The first step in our approach is to introduce a matrix representation of the states. In most applications states are multi-dimensional and as a result, they can be indexed by a tuple of indexes. To keep

the exposition simple, suppose that two indices are used (either because there are only two dimensions, or because the different dimensions are clustered into two groups)¹. With this in mind, define $\mathbf{X}_\mu \in \mathbb{R}^{N \times M}$ as the matrix that collects the means associated with each of the $N_S = N \cdot M$ states. The idea under this approach is that every state $s \in \mathcal{S}$ is coded into two indices $i_s \in \{1, \dots, N\}$ and $j_s \in \{1, \dots, M\}$ and, then, the mean associated with s_t is simply obtained as $\mu(s_t) = [\mathbf{X}_\mu]_{i_{s_t}, j_{s_t}}$. Analogously, we define $\mathbf{X}_\sigma \in \mathbb{R}^{N \times M}$ and set $\sigma(s_t) = [\mathbf{X}_\sigma]_{i_{s_t}, j_{s_t}}$. If no structure is imposed on \mathbf{X}_μ and \mathbf{X}_σ , the estimation of μ and σ is non-parametric and, as a result, the updates in (6) and (7) suffice to estimate the policy.

However, such an approach would suffer from the curse of dimensionality. To avoid this, we impose additional structure on \mathbf{X}_μ and \mathbf{X}_σ , forcing them to be low rank². In particular, this entails introducing the matrices $\mathbf{L}_\mu \in \mathbb{R}^{M \times K}$, $\mathbf{R}_\mu \in \mathbb{R}^{K \times N}$, $\mathbf{L}_\sigma \in \mathbb{R}^{M \times K}$ and $\mathbf{R}_\sigma \in \mathbb{R}^{K \times N}$, and write the mean and standard deviation matrices as $\mathbf{X}_\mu = \mathbf{L}_\mu \mathbf{R}_\mu$ and $\mathbf{X}_\sigma = \mathbf{L}_\sigma \mathbf{R}_\sigma$. This model is still non-parametric, but it helps alleviating the curse of dimensionality. The implications of this design decision are twofold: i) the rank of \mathbf{X}_μ and \mathbf{X}_σ is at most K , limiting the degrees of freedom of those matrices (hence, facilitating its estimation from a limited number of observations); and ii) the parameters to estimate are no longer the entries of \mathbf{X}_μ and \mathbf{X}_σ , but the entries of \mathbf{L}_μ , \mathbf{R}_μ , \mathbf{L}_σ and \mathbf{R}_σ . To see the latter point more clearly, note that the mapping from the state to the parameters of the Gaussian under the low-rank models is given by

$$\mu(s_t) = \sum_{k=1}^K [\mathbf{L}_\mu]_{i_{s_t}, k} [\mathbf{R}_\mu]_{k, j_{s_t}} \mid \sigma(s_t) = \sum_{k=1}^K [\mathbf{L}_\sigma]_{i_{s_t}, k} [\mathbf{R}_\sigma]_{k, j_{s_t}}. \quad (8)$$

The next step is to optimize/estimate the values of the entries of \mathbf{L}_μ , \mathbf{R}_μ , \mathbf{L}_σ and \mathbf{R}_σ . To that end, we resort to non-convex gradient-based matrix factorization approaches [5] along with the expressions for the gradients provided in (6)–(7). To be more specific, in our approach the parameters θ are the entries of $\{\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma\}$ and our goal is to find an expression for the (entries of the) gradient in (3). For simplicity, let us focus first on \mathbf{L}_μ and \mathbf{R}_μ . We aim at finding the expression for the entries of $\nabla_{\mathbf{L}_\mu} J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)$, and $\nabla_{\mathbf{R}_\mu} J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)$. To that end, we need to apply the chain rule and combine the partial derivatives of the cost J w.r.t. μ in (6) with the partial derivatives of the μ function in (8) w.r.t. each entry of the matrices \mathbf{L}_μ , and \mathbf{R}_μ . The result of this is

$$\frac{\partial J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)}{\partial [\mathbf{L}_\mu]_{i, k}} = \sum_{t=0}^T \mathbb{I}_{i=i_{s_t}} A(s_t) \frac{a_t - [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}}}{[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}^2} [\mathbf{R}_\mu]_{k, j_{s_t}} \quad (9)$$

$$\frac{\partial J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)}{\partial [\mathbf{R}_\mu]_{k, j}} = \sum_{t=0}^T \mathbb{I}_{j=j_{s_t}} A(s_t) \frac{a_t - [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}}}{[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}^2} [\mathbf{L}_\mu]_{i_{s_t}, k} \quad (10)$$

where $\mathbb{I}_{i=i_{s_t}}$, and $\mathbb{I}_{j=j_{s_t}}$ are indicator functions. An analogous approach considering the derivatives of the cost J w.r.t. σ in (6), along

¹Generalizations of this approach to (low-rank) tensor models are straightforward and well-motivated, but for simplicity, we limit this conference paper to the matrix case.

²Alternative schemes that *promote* low-rank via, e.g., nuclear norm regularizers are also possible.

with the derivatives of σ w.r.t. the entries of \mathbf{L}_σ and \mathbf{R}_σ yields:

$$\frac{\partial J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)}{\partial [\mathbf{L}_\sigma]_{i, k}} = \sum_{t=0}^T \mathbb{I}_{i=i_{s_t}} A(s_t) \left(\frac{(a_t - [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}})^2}{2[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}^3} - \frac{1}{[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}} \right) [\mathbf{R}_\sigma]_{k, j_{s_t}} \quad (11)$$

$$\frac{\partial J(\mathbf{L}_\mu, \mathbf{R}_\mu, \mathbf{L}_\sigma, \mathbf{R}_\sigma)}{\partial [\mathbf{R}_\sigma]_{k, j}} = \sum_{t=0}^T \mathbb{I}_{j=j_{s_t}} A(s_t) \left(\frac{(a_t - [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}})^2}{2[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}^3} - \frac{1}{[\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}} \right) [\mathbf{L}_\sigma]_{i_{s_t}, k} \quad (12)$$

Once the actor is properly set up, the last step is to define the scheme to estimate ω , the parameters of the critic. As for the actor, we use a matrix representation $\mathbf{X}_\omega \in \mathbb{R}^{N \times M}$ for the VF and then postulate that the VF matrix is low rank, so that it can be factorized as the product of $\mathbf{L}_\omega \in \mathbb{R}^{N \times K}$ and $\mathbf{R}_\omega \in \mathbb{R}^{K \times M}$ with $K \ll \min\{N, M\}$. As a result, the functional expression of the critic takes the form $V(s_t) = \sum_{k=1}^K [\mathbf{L}_\omega]_{i_{s_t}, k} [\mathbf{R}_\omega]_{k, j_{s_t}}$. We then optimize the critic using alternating gradient descent, with the partial derivatives of the critic cost in (4) w.r.t. the entries of \mathbf{L}_ω and \mathbf{R}_ω being

$$\frac{\partial \mathcal{L}(\mathbf{L}_\omega, \mathbf{R}_\omega)}{\partial [\mathbf{L}_\omega]_{i, k}} = \sum_{t=0}^T \mathbb{I}_{i=i_{s_t}} (G_t - [\mathbf{L}_\omega \mathbf{R}_\omega]_{i_{s_t}, j_{s_t}}) [\mathbf{R}_\omega]_{k, j_{s_t}} \quad (13)$$

$$\frac{\partial \mathcal{L}(\mathbf{L}_\omega, \mathbf{R}_\omega)}{\partial [\mathbf{R}_\omega]_{k, j}} = \sum_{t=0}^T \mathbb{I}_{j=j_{s_t}} (G_t - [\mathbf{L}_\omega \mathbf{R}_\omega]_{i_{s_t}, j_{s_t}}) [\mathbf{L}_\omega]_{i_{s_t}, k} \quad (14)$$

The algorithm. A low-rank policy gradient (LRPG) algorithm flows from the definitions above. The agent samples a full trajectory using the Gaussian policy $\pi_{\mu, \sigma} = \mathcal{N}(a_t | \mu(s_t), \sigma(s_t))$, where the mean is $\mu(s_t) = [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}}$, and the standard deviation is $\sigma(s_t) = [\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}$. Then, the actor matrices \mathbf{L}_μ , \mathbf{R}_μ , \mathbf{L}_σ , and \mathbf{R}_σ are updated via stochastic gradient ascent using (9)–(12). Finally, the critic matrices \mathbf{L}_ω , and \mathbf{R}_ω are updated via stochastic gradient descent using (13)–(14). The algorithm is depicted in Algorithm 1.

4. NUMERICAL EXPERIMENTS

We test LRPG in three standard continuous-action RL problems of the toolkit OpenAI Gym [31]. The first scenario is the inverted pendulum, where an agent tries to keep a pendulum upright. The second one is the acrobat, which mimics a gymnast trying to swing up. The third one is the Goddard problem, a rocket optimizing its peak altitude ascending vertically. We test and compare NN-based vs low-rank-based policy methods. The figures of merit reported are i) parametrization-efficiency, ii) convergence rate, and iii) return obtained by the estimated policy. Details can be found in [32].

Experimental setup. We compared the LRPG algorithm against REINFORCE with VF-baselines (RVFB). In both scenarios, given a state s_t , the action a_t is sampled from the normal distribution $\mathcal{N}(a | \mu(s_t), \sigma)$. For the sake of simplicity, σ does not depend on s_t and the difference between the schemes resides in the model to estimate the mean $\mu(s_t)$. While the parameters for LRPG are the entries of \mathbf{L}_μ and \mathbf{R}_μ , the RVFB algorithm uses an NN model $\mu_\theta(s_t) = \text{NN}_\theta(s_t)$. Similarly, LRPG models the VF as $V(s_t) = [\mathbf{L}_\omega \mathbf{R}_\omega]_{i_{s_t}, j_{s_t}}$, and RVFB as $V_\omega(s_t) = \text{NN}_\omega(s_t)$. Since the three setups tested are continuous, LRPG discretizes the state

Algorithm 1 Low Rank Policy Gradient (LRPG)

Require: Initial policy and VF matrices $\mathbf{L}_\mu^0, \mathbf{R}_\mu^0, \mathbf{L}_\sigma^0, \mathbf{R}_\sigma^0, \mathbf{L}_\omega^0$, and \mathbf{R}_ω^0 ; learning rates $\alpha_\mu, \alpha_\sigma, \alpha_\omega$; and maximum number of episodes H .

for $h = 0, \dots, H$ **do**
 Observe initial state s_0
 for $t = 0, \dots, T$ **do** ▷ Sample a trajectory τ_T
 $\mu_{s_t} \leftarrow [\mathbf{L}_\mu \mathbf{R}_\mu]_{i_{s_t}, j_{s_t}}$
 $\sigma_{s_t} \leftarrow [\mathbf{L}_\sigma \mathbf{R}_\sigma]_{i_{s_t}, j_{s_t}}$
 $a_t \sim \mathcal{N}(a | \mu_{s_t}, \sigma_{s_t})$
 Take action a_t , and observe next state $s_{t'}$, and reward r_t
 $s_t \leftarrow s_{t'}$
 end for

 $\mathbf{L}_\mu^{h+1} \leftarrow \mathbf{L}_\mu^h + \alpha_\mu \nabla_{\mathbf{L}_\mu} J(\mathbf{L}_\mu^h, \mathbf{R}_\mu^h, \mathbf{L}_\sigma^h, \mathbf{R}_\sigma^h)$ ▷ Actor update
 $\mathbf{R}_\mu^{h+1} \leftarrow \mathbf{R}_\mu^h + \alpha_\mu \nabla_{\mathbf{R}_\mu} J(\mathbf{L}_\mu^h, \mathbf{R}_\mu^h, \mathbf{L}_\sigma^h, \mathbf{R}_\sigma^h)$
 $\mathbf{L}_\sigma^{h+1} \leftarrow \mathbf{L}_\sigma^h + \alpha_\sigma \nabla_{\mathbf{L}_\sigma} J(\mathbf{L}_\mu^h, \mathbf{R}_\mu^h, \mathbf{L}_\sigma^h, \mathbf{R}_\sigma^h)$
 $\mathbf{R}_\sigma^{h+1} \leftarrow \mathbf{R}_\sigma^h + \alpha_\sigma \nabla_{\mathbf{R}_\sigma} J(\mathbf{L}_\mu^h, \mathbf{R}_\mu^h, \mathbf{L}_\sigma^h, \mathbf{R}_\sigma^h)$

 $\mathbf{L}_\omega^{h+1} \leftarrow \mathbf{L}_\omega^h + \alpha_\omega \nabla_{\mathbf{L}_\omega} J(\mathbf{L}_\mu^h, \mathbf{R}_\mu^h)$ ▷ Critic update
 $\mathbf{R}_\omega^{h+1} \leftarrow \mathbf{R}_\omega^h + \alpha_\omega \nabla_{\mathbf{R}_\omega} J(\mathbf{L}_\omega^h, \mathbf{R}_\omega^h)$
end for

Env.	LRPG		RVFB	
	Parameters	Return	Parameters	Return
Pendulum	160	99.34	4,098	98.81
Acrobot	16	94.45	386	94.25
Rocket	80	1.17	1,282	1.16

Table 1. Parameters vs. Median return.

space \mathcal{S} . Note that the finer the discretization, the larger the number of entries of \mathbf{X}_μ and \mathbf{X}_ω . Imposing low rank can drastically reduce the number of parameters while keeping a fine sampling resolution. The number of total states is defined by the Cartesian product of a regularly-sampled grid. On the other hand, NNs can deal with continuous setups, but choosing the proper architecture is usually a hard task. For the presented scenarios, we run 100 simulations of each algorithm. In each episode, we measure the return per episode $\bar{\mathcal{R}} = \sum_{t=0}^T r_t$. In Fig. 1 the median $\bar{\mathcal{R}}$ (across the 100 simulations) is shown. We have exhaustively examined the space of all potential fully-connected NN architectures to find the smallest one that solves the problem. A summary of the parametrizations and associated median returns is provided in Table 1.

Convergence properties. The rate of convergence of LRPG and RVFB can be observed in Fig. 1. Here, we use the term convergence loosely to refer to the fact that the return reaches a steady state. In all scenarios, LRPG needs fewer episodes to converge to the final policy than RVFB. The strength of this finding varies across scenarios, being conspicuous in some of them. In the Pendulum scenario, LRPG algorithm needs almost 500 episodes less to converge than RVFB (5 times faster). Similarly, in the Goddard problem, LRPG converges around episode 7, 000, while RVFB converges in episode 22, 000. Note, however, that this is not the case in the acrobot setup, where LRPG converges only slightly faster than RVFB. The faster convergence of LRPG is likely due to the fact that low-rank factorized models are simpler and have fewer parameters to estimate (see Table 1). Furthermore, trading off between convergence rate and obtained return is significantly harder in NN-based setups since exploring and testing NN architectures is challenging.

Parametrization efficiency. As summarized in Table 1, LRPG needs far fewer parameters than RVFB to estimate its associated

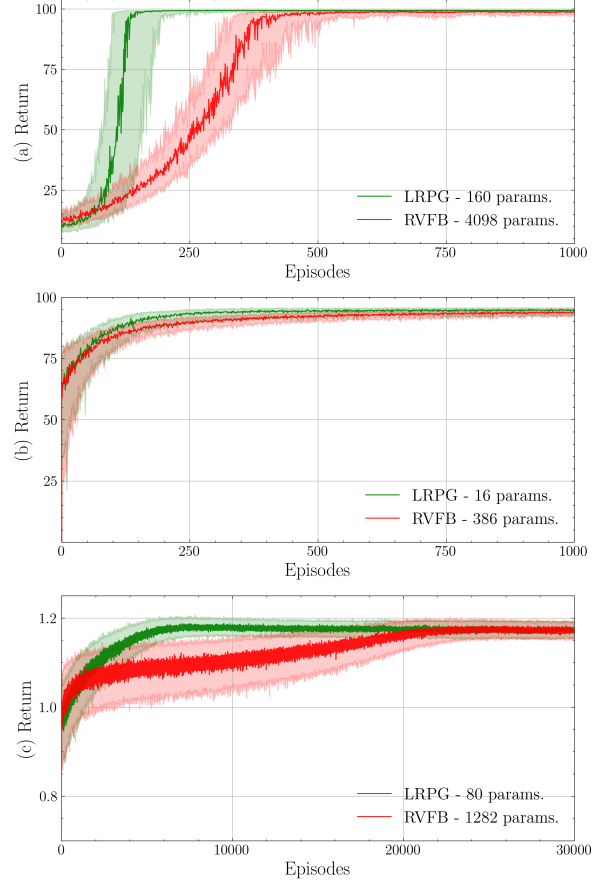


Fig. 1. Median return per episode in three standard RL problems: (a) the pendulum, (b) the acrobot, and (c) the Goddard problem.

policy. In the pendulum and acrobot environments, the size of the LRPG low-rank matrix-based models is approximately 4% of that of the NNs in RVFB. In the case of the Goddard problem, the size of the matrix model is around 6% of the size of the NN model. As mentioned previously, no smaller fully-connected NN was found to converge in these problems. Yet the policies estimated by LRPG lead to higher returns in all scenarios. NNs tend to have problems with local optima, especially in RL setups, and the space of all possible architectures is vast. Hence, the problem of finding the adequate architecture not only affects the reward, but also the convergence properties of the algorithm. In contrast, tuning the LRPG algorithm boils down to designing the discretization and indexing of the state space \mathcal{S} , along with the rank of the model K .

5. CONCLUSIONS

This paper introduced a *low-rank* policy gradient (LRPG) algorithm, which leverages matrix completion in (discrete) matrix-based Gaussian policy models. This approach portrays a natural way of introducing parsimony in policy-based RL techniques. The low-rank policy is very efficient in terms of parameters, and it is easy to interpret. LRPG was tested in three OpenAI Gym classical control environments, comparing it with standard NN-based schemes. The limited but meaningful tests reveal that LRPG achieves high rewards, usually better than NN-based models, with a relatively small number of parameters. Furthermore, it requires significantly less number of episodes to converge to the final policy.

6. REFERENCES

- [1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2nd Ed., 2018.
- [2] D. P. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.
- [3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [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] 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.
- [7] 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.
- [8] E. Tolstaya, A. Koppel, E. Stump, and A. Ribeiro, “Nonparametric stochastic compositional gradient descent for q-learning in continuous markov decision problems,” in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 6608–6615.
- [9] G. Lever, J. Shave-Taylor, R. Stafford, and C. Szepesvári, “Compressed conditional mean embeddings for model-based reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.
- [10] 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.
- [11] 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.
- [12] A. Mahajan, M. Samvelyan, L. Mao, V. Makoviychuk, A. Garg, J. Kossaifi, S. Whiteson, Y. Zhu, and A. Anandkumar, “Tesseract: Tensorised actors for multi-agent reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 7301–7312.
- [13] H. Y. Ong, “Value function approximation via low-rank models,” *arXiv preprint arXiv:1509.00061*, 2015.
- [14] 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.
- [15] 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.
- [16] F. S. Melo and M. I. Ribeiro, “Q-learning with linear function approximation,” in *Intl. Conf. Comp. Learning Theory*. Springer, 2007, pp. 308–322.
- [17] 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.
- [18] Y. Yang, G. Zhang, Z. Xu, and D. Katabi, “Harnessing structures for value-based planning and reinforcement learning,” in *International Conference on Learning Representations*, 2020.
- [19] D. Shah, D. Song, Z. Xu, and Y. Yang, “Sample efficient reinforcement learning via low-rank matrix estimation,” in *Proc. of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS’20. Red Hook, NY, USA: Curran Associates Inc., 2020.
- [20] S. Rozada, V. Tenorio, and A. G. Marques, “Low-rank state-action value-function approximation,” in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 1471–1475.
- [21] S. Rozada and A. G. Marques, “Tensor and matrix low-rank value-function approximation in reinforcement learning,” *arXiv preprint arXiv:2201.09736*, 2022.
- [22] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [23] 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.
- [24] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.
- [25] J. Peters and S. Schaal, “Natural actor-critic,” *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.
- [26] D. Lee, N. He, P. Kamalaruban, and V. Cevher, “Optimization for reinforcement learning: From a single agent to cooperative agents,” *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 123–135, 2020.
- [27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Advances in neural information processing systems*, vol. 12, 1999.
- [28] E. Greensmith, P. L. Bartlett, and J. Baxter, “Variance reduction techniques for gradient estimates in reinforcement learning,” *Journal of Machine Learning Research*, vol. 5, no. 9, 2004.
- [29] S. Levine and P. Abbeel, “Learning neural network policies with guided policy search under unknown dynamics,” *Advances in neural information processing systems*, vol. 27, 2014.
- [30] K. Ciosek and S. Whiteson, “Expected policy gradients,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [32] S. Rozada, “Online code repository: Matrix low-rank approximation for policy gradient methods,” <https://github.com/sergiorozada12/matrix-low-rank-pg>, 2022.