An Empirical Analysis of Measure-Valued Derivatives for Policy Gradients

João Carvalho[1], Davide Tateo[1], Fabio Muratore[1,2], Jan Peters[1,3]

Intelligent Autonomous Systems
Technische Universität Darmstadt, Darmstadt, Germany
{joao,davide,fabio}@robot-learning.de, jan.peters@tu-darmstadt.de

Abstract—**Reinforcement learning methods for robotics are increasingly successful due to the constant development of better policy gradient techniques. A *precise* (low variance) and *accurate* (low bias) gradient estimator is crucial to face increasingly complex tasks. Traditional policy gradient algorithms use the likelihood-ratio trick, which is known to produce unbiased but high variance estimates. More modern approaches exploit the reparametrization trick, which gives lower variance gradient estimates but requires differentiable value function approximators. In this work, we study a different type of stochastic gradient estimator: the Measure-Valued Derivative. This estimator is unbiased, has low variance, and can be used with differentiable and non-differentiable function approximators. We empirically evaluate this estimator in the actor-critic policy gradient setting and show that it can reach comparable performance with methods based on the likelihood-ratio or reparametrization tricks, both in low and high-dimensional action spaces.**

I. INTRODUCTION

Complex robotics tasks, such as locomotion and manipulation, can be formulated as Reinforcement Learning (RL) problems in continuous state and action spaces [1]. The RL objective is commonly an expectation dependent on a parameterized control policy, whose optimization is done by computing a gradient w.r.t. the policy parameters to steadily move them in the gradient ascent direction [2] – known as the policy gradient. If the control policy is stochastic, this method can be viewed as computing the gradient of an expectation w.r.t. a distribution containing the policy parameters, both in the actor-only version [3], as well as in the actor-critic one [4].

If the gradient of the expectation of a function w.r.t. the distribution parameters cannot be found analytically, there are three main approaches to obtain an unbiased estimate: the likelihood-ratio, also called Score-function (SF); the Pathwise Derivative, commonly known as the Reparametrization trick (Rep-trick); and the Measure-Valued Derivative (MVD) [5].

In policy gradients, the gradient estimation problem has been solved extensively using the SF, which is known to pro-

duce high-variance estimates, particularly if used in trajectory-based formulations, since its variance grows linearly with the number of steps in the trajectory [6]. Thus, many methods require a baseline for variance reduction [7], and the optimal one needs to be computed for every single case [8]. Frequently used baseline heuristics are the mean of the returns or the state value function. REINFORCE is an example of a SF based method [3].

Generally, the Rep-trick [9] is a low variance gradient estimator that can be used in distributions of continuous and discrete random variables. For the latter, the Gumbel-softmax trick allows the backpropagation of gradients [10]. Moreover, it requires the function we are optimizing to be differentiable, which excludes non-differentiable approximators, such as regression trees [11]. Although the Rep-trick often produces low variance estimates, this property does not hold for all function classes, in particular those whose derivative has a high Lipschitz constant [12]. An example of an algorithm using this estimator is the Soft Actor-Critic (SAC) [13].

MVD is another method to compute unbiased gradient estimates, which has close connections to Finite-difference (FD), and it is not used often in the Machine Learning (ML) community. Similarly to the SF it applies to both discrete and continuous distributions, but it generally has low variance and avoids computing an extra baseline for variance reduction. Contrary to the Rep-trick, it is not restricted to differentiable functions, which makes it applicable to more function classes. Few works have studied its application to RL [6], [14].

In this paper, we analyze the properties of MVD in actor-critic policy gradient algorithms for on and off-policy RL. We compare the estimation errors produced by the three different estimators in Linear-Quadratic Regulator (LQR) problems, for which we know the true value function and policy gradient. We show that replacing the Rep-trick with MVD in an off-policy deep RL algorithm leads to similar performance results, even in complex environments with high-dimensional state and action spaces. To argue that MVDs can be generally used with broader classes of critic approximators, where the Rep-trick is not available, we construct an example of an on-policy procedure that uses a Q-function fitted with Extra-Trees [11] – a non-differentiable model. The results suggest that this type of estimator can be used to compute policy gradients in continuous state and action spaces.

II. RELATED WORK

Policy gradient methods can be derived through the trajectory, also called actor-only, or actor-critic formulations [2]. In the former, the expected return from a state-action pair is estimated with samples from the trajectory rollout, as done in REINFORCE [3] and GPOMDP [7]. In stochastic environments this results in high variance gradient estimates and poor convergence. Instead, an actor-critic agent performs updates by estimating the returns using a state-action value function approximator. These methods have lower variance and show more stable learning. Several actor-critic algorithms have been presented in the recent years, particularly using deep neural networks as function approximators, including Trust Region Policy Optimization (TRPO) [15], Deep Deterministic Policy Gradient (DDPG) [16], Twin Delayed DDPG (TD3) [17] and SAC [13]. The connection between trajectory and actor-critic methods was established in the policy gradient theorem in its stochastic [4] and deterministic versions [18].

Most policy gradient methods optimize stochastic policies using the SF or Rep-trick, while MVDs have not yet been extensively explored in the ML and RL communities. A modified version of the policy gradient theorem using MVDs, also referred to as weak derivatives, was first introduced in [6]. The authors derive an unbiased estimator and provide an extensive theoretical analysis of its properties, including the variance and computational complexity. However, the practical implementation of the algorithm has to perform two rollouts starting from the same state to obtain an unbiased estimate of the state-action value function. Applying this approach to a real-world scenario, where we cannot reset a system to an arbitrary continuous state, is impossible. Furthermore, it makes use of Monte Carlo rollouts to estimate the return from a state-action pair, discarding all the collected data after an update, which leads to inefficient use of samples. Other works used MVDs to solve Constrained Markov Decision Processes [14], [19], also assuming that the environment can be reset to any state. To the best of our knowledge, there is no actor-critic algorithm based on MVDs that replicates a RL scenario.

III. BACKGROUND

A. Reinforcement Learning and Policy Gradients

Let a Markov Decision Process (MDP) be defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma, \mu_0)$, where $\mathcal{S}$ is a continuous state space $s \in \mathcal{S}$, $\mathcal{A}$ is a continuous action space $a \in \mathcal{A}$, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a transition probability function, with $\mathcal{P}(s'|s, a)$ the density of landing in state s' when taking action a in state s, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1)$ is a discount factor, and $\mu_0 : \mathcal{S} \to \mathbb{R}$ the initial state distribution. A policy π is a mapping from states to actions. If deterministic, it defines what action to take in each state $a = \pi(s)$, while a stochastic one assigns a probability distribution over possible

actions $a \sim \pi(\cdot|s)$. The discounted state visitation under a policy π is defined as $d_\gamma^\pi(s) = \sum_{t=0}^\infty \gamma^t P(s_t = s|s_0, \pi)$, where s_0 is the initial state, and $P : \mathcal{S} \to \mathbb{R}$ the probability of being in state s at time step t given the initial state and the policy. The discounted state distribution is then given by $\mu_\gamma^\pi(s) = (1 - \gamma)d_\gamma^\pi(s)$. The state-action value function – Q-function – is the discounted sum of rewards collected from a given state-action pair following the policy π, $Q^\pi(s, a) = \mathbb{E}_{\pi, \mathcal{P}}[\sum_{t=0}^\infty \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a]$, and the state value function is its expectation w.r.t. the policy $V^\pi(s) = \mathbb{E}_{a \sim \pi}[Q^\pi(s, a)]$. The advantage function is given as the difference between the two $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. In general, the goal of a RL agent is to maximize the expected sum of discounted rewards from any initial state

$$J(\pi) = \mathbb{E}_{s_0 \sim \mu_0}[V^\pi(s_0)] = \mathbb{E}_{\tau \sim \mu_0, \pi, \mathcal{P}}\left[\sum_{t=0}^\infty \gamma^t r_t\right], \quad (1)$$

where τ is a state-action trajectory $(s_0, a_0, s_1, a_1, \dots)$, determined by the policy and the environment dynamics.

In high-dimensional and continuous action spaces, instead of constructing value functions and retrieving optimal actions afterwards, a policy with parameters θ is updated iteratively with a gradient ascent step on (1), $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$. The on-policy policy gradient theorem [4] establishes this gradient for stochastic policies as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s \sim \mu_\gamma^\pi}\left[\int \nabla_\theta \pi(a|s; \theta)Q^\pi(s, a)\, \mathrm{d}a\right], \quad (2)$$

which can be extended to the off-policy [20] and deterministic policy [18] settings. Note that the integral in (2) is not an expectation.

B. Monte Carlo Gradient Estimators

The optimization goal of several problems in ML, e.g. Variational Inference (VI), is often posed as an expectation of a function f w.r.t. a distribution p parameterized by ω

$$J(\omega) = \mathbb{E}_{p(x; \omega)}[f(x)] = \int p(x; \omega)f(x)\, \mathrm{d}x, \quad (3)$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function of $x \in \mathbb{R}^n$, $p : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the distribution of x, and $\omega \in \mathbb{R}^m$ are the parameters encoding the distribution, also known as distributional parameters. For instance, in amortized VI [9], f is the Evidence Lower Bound (ELBO) and p the approximate posterior distribution. If $p(x; \omega)$ is a multivariate Gaussian distribution over x, then ω aggregates the mean and covariance.

A gradient ascent algorithm is a common choice to find the parameters ω that maximize (3). For that we need to compute the gradient

$$\nabla_\omega J(\omega) = \int \nabla_\omega p(x; \omega)f(x)\, \mathrm{d}x, \quad (4)$$

where for clarity of explanation we assume interchangeability of differentiation and integration (see [5] for cases where it does not apply). Since the derivative of a distribution is in general not a distribution itself, the integral in (4) is not an

expectation and thus not solvable directly by Monte Carlo (MC) sampling.

If f also depends on $\boldsymbol{\omega}$, the product rule gives $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \int \nabla_{\boldsymbol{\omega}} p(\boldsymbol{x};\boldsymbol{\omega}) f(\boldsymbol{x};\boldsymbol{\omega}) \,\mathrm{d}\boldsymbol{x} + \int p(\boldsymbol{x};\boldsymbol{\omega}) \nabla_{\boldsymbol{\omega}} f(\boldsymbol{x};\boldsymbol{\omega}) \,\mathrm{d}\boldsymbol{x}$. The second term in the sum is an expectation, and can thus be computed by sampling. Hence, to keep the notation simple, we assume f does not depend on $\boldsymbol{\omega}$, as its extension is trivial.

In many problems we do not have access to the true function f, e.g. value functions in RL, and need to use approximators $\hat{f}$, which introduce bias and variance. Therefore we are typically interested in finding unbiased estimators for (4), to prevent the accumulation of the estimation errors already given by $\hat{f}$.

Let $\hat{\boldsymbol{g}}_i \in \mathbb{R}^m$ define one unbiased gradient estimate of (4). Then a MC estimation of the gradient using M samples is obtained as $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) \approx 1/M \sum_{i=1}^{M} \hat{\boldsymbol{g}}_i$. A straightforward variance reduction technique common to any method is to use more samples, as the variance reduces with $\mathcal{O}(M^{-1})$. However, this increases the computational complexity.

Next, we summarize the three known ways to build an unbiased estimator for (4). See [5] for a throughout analysis.

a) Reparametrization trick (Rep-trick) [9]: A random variable $\boldsymbol{x}$ is reparametrizable if it can be obtained by constructing a deterministic path from a base random variable $\boldsymbol{\varepsilon}$ by applying a deterministic function g with parameters $\boldsymbol{\omega}$, such that $\boldsymbol{x} = g(\boldsymbol{\varepsilon};\boldsymbol{\omega})$. By the change of variables we have $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \int p(\boldsymbol{\varepsilon}) \nabla_{\boldsymbol{\omega}} f(g(\boldsymbol{\varepsilon};\boldsymbol{\omega})) \,\mathrm{d}\boldsymbol{\varepsilon}$. A single gradient estimate is given as $\hat{\boldsymbol{g}}_i^{\mathrm{Rep}} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x} = g(\boldsymbol{\varepsilon}_i;\boldsymbol{\omega})) \nabla_{\boldsymbol{\omega}} g(\boldsymbol{\varepsilon}_i;\boldsymbol{\omega})$, with $\boldsymbol{\varepsilon}_i \sim p(\boldsymbol{\varepsilon})$. This estimator requires f to be differentiable w.r.t. $\boldsymbol{x}$, to be able to compute its derivative, and typically leads to low-variance estimates. However, it can have high variance if f is rough [21].

b) Score-function (SF) [22]: From the derivative of the logarithm identity we rewrite (4) as $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{\omega}) = \int p(\boldsymbol{x};\boldsymbol{\omega}) \nabla_{\boldsymbol{\omega}} \log p(\boldsymbol{x};\boldsymbol{\omega}) f(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}$. A single gradient estimate is given as $\hat{\boldsymbol{g}}_i^{\mathrm{SF}} = \nabla_{\boldsymbol{\omega}} \log p(\boldsymbol{x}_i;\boldsymbol{\omega}) f(\boldsymbol{x}_i)$, with $\boldsymbol{x}_i \sim p(\boldsymbol{x};\boldsymbol{\omega})$. Contrary to the Rep-trick, it does not need the derivative of f, but just that we can query it. It is common for the estimates to have large variance, especially in a stochastic process, as it grows linearly with the number of steps [19]. In practice, the variance is reduced by subtracting a baseline to f, keeping the estimator unbiased. Several policy gradient algorithms are based on this estimator [3], [7], [23].

c) Measure-Valued Derivative (MVD) [24]: Even though the derivative of a distribution is not in general a distribution, it is a difference between distributions up to a normalizing constant [24]. The main idea behind MVD is to write the derivative w.r.t. a single distributional parameter $\omega_k \in \mathbb{R}$ as a difference between two distributions, i.e. $\nabla_{\omega_k} p(\boldsymbol{x};\boldsymbol{\omega}) = c_{\omega_k} \left(p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega}) - p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega}) \right)$, where c_{ω_k} is a normalizing constant that can depend on ω_k, and $p_{\omega_k}^+$ and $p_{\omega_k}^-$ are two distributions referred to as the positive and negative components, respectively. The MVD is described by the triplet $\left(c_{\omega_k}, p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega}), p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega}) \right)$. The decomposition is not unique, and different configurations of the triplet can be obtained, which leads to estimators with different properties.

For common distributions, such as the Gaussian, Poisson, or Gamma, decompositions have already been analytically derived in [25]. Table I shows a common one used for the mean and standard deviation of the Gaussian distribution.

Expanding (4) with the MVD decomposition we obtain $\nabla_{\omega_k} J(\boldsymbol{\omega}) = \int c_{\omega_k} \left(p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega}) - p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega}) \right) f(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}$. A single gradient estimate for ω_k is given by $\hat{g}_{k,i}^{\mathrm{MVD}} = c_{\omega_k} \left(f(\boldsymbol{x}^+;\boldsymbol{\omega}) - f(\boldsymbol{x}^-;\boldsymbol{\omega}) \right)$, with $\boldsymbol{x}^+ \sim p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega})$ and $\boldsymbol{x}^- \sim p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega})$. I.e., the derivative is the difference between the function f evaluated at different points sampled according to the decomposition, and scaled with the normalization constant. For example, to compute the derivative of a one-dimensional Gaussian $\mathcal{N}(x;\mu,\sigma)$ w.r.t. its standard deviation, we can sample the positive part from a Double-sided Maxwell and the negative one from a Gaussian. The gradient w.r.t. all parameters is just the concatenation of the derivatives w.r.t. all single parameters ω_k. Computing one gradient estimate for all parameters $\boldsymbol{\omega}$ is in the order of $\mathcal{O}(2|\boldsymbol{\omega}|)$ queries to f, in contrast to the SF and Rep-trick $\mathcal{O}(1)$ complexity.

Let $\hat{g}_k^{\mathrm{MVD}}$ be the estimator of $\nabla_{\omega_k} \mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\omega})} [f(\boldsymbol{x})]$, with variance given by [5]

$$\mathbb{V}_{p(\boldsymbol{x};\boldsymbol{\omega})} \left[\hat{g}_k^{\mathrm{MVD}} \right] = \mathbb{V}_{p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega})} [f(\boldsymbol{x})] + \mathbb{V}_{p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega})} [f(\boldsymbol{x})] - 2\mathrm{Cov}_{p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega}) p_{\omega_k}^-(\boldsymbol{x}';\boldsymbol{\omega})} [f(\boldsymbol{x}), f(\boldsymbol{x}')],$$

which depends on the chosen decomposition and how correlated are the function evaluations at the positive and negative samples. A typical choice for variance reduction is to obtain a decomposition such that the positive and negative distributions are orthogonal [26], such as for the Gaussian mean in Table I. Moreover, if $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ are positively correlated, then the covariance term is positive and the variance is further reduced. This last term is possibly reduced through *coupling*, which works by constructing a variate from the negative component $\boldsymbol{x}'$ by applying a transformation to the sample from the positive component (or vice-versa), e.g. by using common random numbers, such that they share the same source of randomness. A simple example is when computing the gradient w.r.t. the mean of a Gaussian: first we sample a variate from the Weibull distribution $\mathcal{W}(\sqrt{2},2)$ to construct the positive component with $\mu + \sigma\mathcal{W}(\sqrt{2},2)$, and then reuse the same variate to also construct the negative component [27]. Even though this does not guarantee that the function evaluations are positively correlated, it tends to work well in practice [5].

If $p(\boldsymbol{x};\boldsymbol{\omega})$ is a multivariate distribution with independent dimensions it factorizes as $p(\boldsymbol{x};\boldsymbol{\omega}) = \prod_{i=1}^{n} p(x_i;\boldsymbol{\xi}_i)$, with $x_i \in \mathbb{R}$ and $\boldsymbol{\xi}_i$ a subset of $\boldsymbol{\omega}$. For instance, if $p(\boldsymbol{x};\boldsymbol{\omega})$ is a Gaussian with diagonal covariance, then $\boldsymbol{\xi}_i = \{\mu_i, \sigma_i\}$. The derivative w.r.t. one parameter $\omega_k \in \boldsymbol{\xi}_k$ is given by

$$\frac{\partial}{\partial \omega_k} p(\boldsymbol{x};\boldsymbol{\omega}) = \prod_{i=1}^{k-1} p(x_i;\boldsymbol{\xi}_i) \frac{\partial}{\partial \omega_k} p(x_k;\boldsymbol{\xi}_k) \prod_{j=k+1}^{n} p(x_j;\boldsymbol{\xi}_j)$$

$$= c_{\omega_k} \left(p_{\omega_k}^+(x_k;\boldsymbol{\xi}_k) - p_{\omega_k}^-(x_k;\boldsymbol{\xi}_k) \right) \prod_{i=1,i\neq k}^{n} p(x_i;\boldsymbol{\xi}_i)$$

$$= c_{\omega_k} \left(p_{\omega_k}^+(\boldsymbol{x};\boldsymbol{\omega}) - p_{\omega_k}^-(\boldsymbol{x};\boldsymbol{\omega}) \right),$$

MVD Gaussian distribution $\mathcal{N}(x;\mu,\sigma),\omega_k=\{\mu,\sigma\}$				Univariate distributions		
$\frac{\partial}{\partial\omega_k}\mathcal{N}$	c_{ω_k}	$p^+_{\omega_k}(x;\omega_k)$	$p^-_{\omega_k}(x;\omega_k)$	Gaussian	$\mathcal{N}(x\in\mathbb{R};\mu,\sigma)$	$\frac{1}{\sqrt{2\pi}\sigma}\exp-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$
$\frac{\partial}{\partial\mu}\mathcal{N}$	$\frac{1}{\sigma\sqrt{2\pi}}$	$\mathcal{W}(x-\mu;\sqrt{2}\sigma,2)$	$\mathcal{W}(\mu-x;\sqrt{2}\sigma,2)$	Weibull	$\mathcal{W}(x\in\mathbb{R}^+;\lambda,k)$	$\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}\exp-\left(\frac{x}{\lambda}\right)^k$
$\frac{\partial}{\partial\sigma}\mathcal{N}$	$\frac{1}{\sigma}$	$\mathcal{M}(x;\mu,\sigma)$	$\mathcal{N}(x;\mu,\sigma)$	Double-sided Maxwell	$\mathcal{M}(x\in\mathbb{R};\mu,\sigma)$	$\frac{1}{\sqrt{2\pi}\sigma^3}(x-\mu)^2\exp-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$

TABLE I: MVD decomposition for the derivative of the Gaussian distribution w.r.t. to the mean and standard deviation (left) and the univariate distributions from the decomposition (right).

where $p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})$ is the original multivariate distribution with the k-th component replaced by the positive part of the decomposition of the univariate marginal $p(x_k;\boldsymbol{\xi}_k)$ w.r.t. ω_k. The negative component $p^-_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})$ is built similarly. A practical algorithm to sample from $p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})$ first samples from the initial multivariate distribution $p(\boldsymbol{x};\boldsymbol{\omega})$ and then replaces the k-th component with a sample from the marginal $p^+_{\omega_k}(x_k;\boldsymbol{\xi}_k)$. E.g., for the derivative w.r.t. the mean of the multivariate Gaussian with diagonal covariance, a sample from $p^+_{\omega_k}(\boldsymbol{x};\boldsymbol{\omega})$ includes sampling first from the original multivariate Gaussian and then replacing the k-entry with a sample from a Weibull.

MVDs are closely related with Finite-difference (FD) methods, as both use perturbations per parameter dimension and take the difference between function evaluations to compute a gradient estimate. To produce an estimator of (4) for each parameter, FD methods construct two distributions by perturbing the distributional parameter ω_k by small amounts $\pm\Delta$, and then sample from the resulting modified distributions. Although the bias disappears for $\Delta \to 0$, the variance grows with $\mathcal{O}(\Delta^{-2})$, making the estimator unreliable [28]. Instead, MVDs can be seen as a principled way to choose the distributions to sample from. Note that the FD is applied to the distributional parameters in (3) and not to the deterministic function f, and thus different from computing gradients to optimize f only.

Illustrative Example: Fig. 1 shows the result of optimizing the expectation of common test functions under a 2-

(a) Quadratic (b) Himmelblau (c) Styblinski
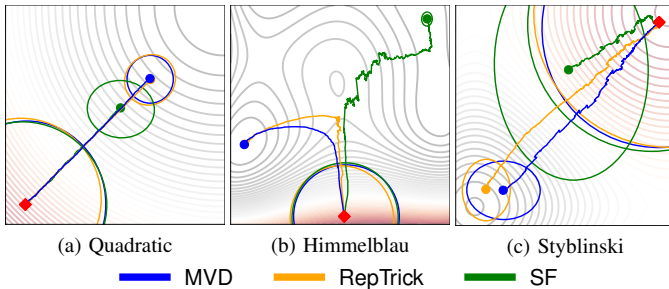
—— MVD   —— RepTrick   —— SF

Fig. 1: Sample run of the unbiased stochastic gradient estimators in optimization test functions for a fixed number of iterations. The lines show the means of the distributions, and the ellipses one standard deviation at the beginning (red diamond) and end (circle). Coloring: lower values in red, higher values in grey.

dimensional Gaussian distribution with diagonal covariance, using gradient ascent with the three referred methods. The SF uses the optimal baseline for black-box optimization based on the Policy Gradients with Parameter based Exploration (PGPE) algorithm [29] to further reduce variance. Rep-trick and MVD consistently move towards the global maximum in all test functions, while the SF shows unstable behaviors. The comparison with the FD method is not included since it is biased for finite Δ, and for $\Delta \to 0$ it showed large variance.

C. Parametrization of distributional parameters

Distributional parameters $\boldsymbol{\omega}(\boldsymbol{\theta})$ can be parameterized further by other parameters $\boldsymbol{\theta}$. Consider for instance that the distribution p represents a Gaussian stochastic policy with fixed covariance, and mean as the output of a neural network with state as input and parameters $\boldsymbol{\theta} \in \mathbb{R}^M$, $p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega}(\boldsymbol{\theta})) = \mathcal{N}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega}=\{\boldsymbol{\mu}(\boldsymbol{s};\boldsymbol{\theta})\},\boldsymbol{\Sigma})$, with $\boldsymbol{a} \in \mathbb{R}^{D_a}$ and $\boldsymbol{s} \in \mathbb{R}^{D_s}$. Hence, $|\boldsymbol{\omega}| = D_a$. Applying the chain rule, the gradient of (3) w.r.t. $\boldsymbol{\theta}$ becomes

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\omega}(\boldsymbol{\theta})) = \int \nabla_{\boldsymbol{\omega}} p(\boldsymbol{x};\boldsymbol{\omega}) f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \nabla_{\boldsymbol{\theta}} \boldsymbol{\omega}(\boldsymbol{\theta}), \quad (5)$$

which allows us to decompose the gradient in two parts – the gradient w.r.t. the distributional parameters $\boldsymbol{\omega}$ and the gradient w.r.t. the parameters $\boldsymbol{\theta}$. In the previous example we obtain $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega}(\boldsymbol{\theta})) = \nabla_{\boldsymbol{\mu}} \mathcal{N}(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\mu},\boldsymbol{\Sigma}) \nabla_{\boldsymbol{\theta}} \boldsymbol{\mu}(\boldsymbol{s};\boldsymbol{\theta})$. In practice, we can have a large neural network encoding the mean, but a tractable action space if $D_a \ll M$. Despite the complexity of MVDs, this fact still makes them tractable for policies with a large number of parameters, but considerable action space.

IV. ACTOR-CRITIC POLICY GRADIENT WITH MVDS

The integral in (2) is an application of (4) to RL, as both are solving the same problem of computing an unbiased stochastic gradient estimate. Hence, given the properties of MVD, especially its low variance, we propose to analyze it in this setting. In RL it is common to only have access to an approximation of the true Q-function, which introduces extra bias and variance. Therefore, it is crucial to have a low variance estimator for the gradient of the expected return.

A straight forward application of MVDs would be to perform black-box optimization to optimize the distributional parameters of an upper-level policy that models the distribution of a (typically deterministic) lower-level policy with parameters $\boldsymbol{\theta} \in \mathbb{R}^M$ [2]. In this case, $|\boldsymbol{\omega}| \geq \mathcal{O}(M)$, and the

Policy gradient theorem	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim \mu_\gamma^\pi} \left[\int \nabla_{\omega_k} \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega}) Q^\pi(\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{a} \right]$
Score-Function	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim \mu_\gamma^\pi, \, \boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s};\boldsymbol{\omega})} \left[\nabla_{\omega_k} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega}) Q^\pi(\boldsymbol{s},\boldsymbol{a}) \right]$
Reparametrization Trick	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim \mu_\gamma^\pi, \, \boldsymbol{\varepsilon} \sim p_{\boldsymbol{\varepsilon}}} \left[\nabla_{\boldsymbol{a}} Q^\pi(\boldsymbol{s},\boldsymbol{a} = h(\boldsymbol{s},\boldsymbol{\varepsilon};\boldsymbol{\omega})) \nabla_{\omega_k} h(\boldsymbol{s},\boldsymbol{\varepsilon};\boldsymbol{\omega}) \right]$
Measure-Valued Derivative	$\nabla_{\omega_k} J(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim \mu_\gamma^\pi} \left[c_{\omega_k} \left(\mathbb{E}_{\boldsymbol{a} \sim \pi_{\omega_k}^+(\cdot|\boldsymbol{s};\boldsymbol{\omega})} [Q^\pi(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\boldsymbol{a} \sim \pi_{\omega_k}^-(\cdot|\boldsymbol{s};\boldsymbol{\omega})} [Q^\pi(\boldsymbol{s},\boldsymbol{a})] \right) \right]$

TABLE II: Policy gradient theorem with the different unbiased stochastic gradient estimators.

computational complexity discourages its application if M is large. In contrast, the SF complexity would be $\mathcal{O}(1)$.

MVDs are also applicable in step-based actor-critic policy gradient methods with stochastic policies. If an approximator for the critic is fitted from collected samples, policy updates are done without further interaction with the system, with queries to compute the gradient update done to the Q-function approximator and not estimated from the real environment. This is in contrast with previous work in [6], where the policy gradient contribution for each state is computed by building an unbiased estimate of the Q-function for two different actions, by performing MC rollouts. We stress that this assumes we can perform two rollouts from the same state. While it can be true when using a simulator, this is not generally applicable to RL, where we assume the agent cannot reset to an arbitrary state, especially in real-world environments. Furthermore, if the model of the environment is available, planning algorithms can be more efficient than RL in solving a task. Hence, applying MVDs in actor-critic settings could be an interesting research direction.

Given a stochastic policy $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\omega} = g(\boldsymbol{s};\boldsymbol{\theta}))$, where $\boldsymbol{\omega}$ are distributional parameters resulting from applying a function g with parameters $\boldsymbol{\theta}$ to the state $\boldsymbol{s}$, for instance neural networks that output the mean and covariance of a Gaussian distribution, the gradient w.r.t. $\boldsymbol{\theta}$ can be easily computed if g is a continuous deterministic function, as in (5). From now on, we only consider the gradient w.r.t. $\boldsymbol{\omega}$.

The policy gradient theorem [4] for a single parameter ω_k can thus be written with the three different estimators as shown in Table II. The MVD formulation of the policy gradient shows that to compute the gradient of one distributional parameter we can sample from the discounted state distribution by interacting with the environment, and then evaluate the Q-function at actions sampled from the positive and negative components of the policy decomposition conditioned on the sampled states, $\pi_{\omega_k}^+(\cdot|\boldsymbol{s};\boldsymbol{\omega})$ and $\pi_{\omega_k}^-(\cdot|\boldsymbol{s};\boldsymbol{\omega})$, respectively. Importantly, the Q-function estimate is the one from the policy π and not from the ones resulting from the positive or negative decompositions. We need a model for Q that we can query without MC sampling because we have to evaluate it for two actions starting from the same state. The function approximator for Q can be a differentiable one, such as a neural network, as done in DDPG or SAC, or a non-differentiable one, such as a regression tree.

Due to the high variance of the SF estimator, the Q-function is replaced by the advantage function, which includes the value function as a baseline for variance reduction, while keeping the estimator unbiased. The benefit of MVD over SF is that they do not need to estimate an extra value function, which can be an extra source of bias and variance. Nevertheless, methods such as Generalized Advantage Estimation (GAE) [30] estimate the advantage function by only approximating the state value function, although introducing bias.

V. GRADIENT ANALYSIS IN THE LQR

The LQR is a well-studied problem in control theory. Here we consider the discounted infinite-horizon discrete-time LQR under a Gaussian stochastic policy $\boldsymbol{a} \sim \mathcal{N}(\cdot|-\mathbf{K}\boldsymbol{s}, \boldsymbol{\Sigma})$, where $\mathbf{K}$ is a learnable feedback gain matrix, and $\boldsymbol{\Sigma}$ a fixed covariance. As the value function expression is known and can be computed by numerically solving the Algebraic Riccati Equation, as well as the gradient (2) w.r.t. $\mathbf{K}$, the LQR is a good baseline for policy gradient algorithms.

In this analysis we construct LQRs, with different state and action dimensions, with dynamics such that the uncontrolled systems are unstable, and then select a suboptimal initial gain $\mathbf{K}_{\text{init}}$ such that the closed-loop is stable. All environments have a fixed initial state. We analyze the policy gradient with the estimators from Table II, computed at the initial gain matrix and state. In the SF the Q-function is replaced by the advantage function for variance reduction. The expectation of the on-policy state distribution is sampled directly from the environment, but the expectation over actions uses critics Q and V as the true value functions from the LQR. This is an idealized scenario to analyze the estimators' variances when there are no errors in the function approximator.

We compare two sources of error. The *relative absolute error* relates the norms of the estimated $\hat{\boldsymbol{g}}$ and true gradient $\boldsymbol{g}$ as $(|\|\hat{\boldsymbol{g}}\| - \|\boldsymbol{g}\||)/\|\boldsymbol{g}\|$, i.e. 0 indicates they have the same magnitude, and the *cosine distance* as the error in the gradient direction as $1 - \hat{\boldsymbol{g}}^\intercal \boldsymbol{g}/(\|\hat{\boldsymbol{g}}\|\|\boldsymbol{g}\|)$, i.e. 0 indicates collinearity. An ideal estimator has both measures close to zero.

The errors are analyzed along two dimensions – the number of trajectories and the number of actions sampled to solve the action expectations. Using MVD the gradient w.r.t. the mean needs two function evaluations per action dimension. For a fair comparison, the Rep-trick and SF sample the same number of actions the MVD needs for the gradient of all parameters. This way the computation complexity is not exactly the same,

since for the Rep-trick we still need to compute the gradient of Q and for SF the log-probability derivative, but gives a fair comparison than just using one MC estimate. The results are shown in Fig. 2. As expected, with the number of trajectories fixed, increasing the number of sampled actions decreases the estimators' errors. The Rep-trick achieves the best results in

both magnitude and direction in all environments, and the MVD is slightly better than the SF. In our experiments we observed this holds for higher dimensions as well. This reveals that even though the SF complexity is $\mathcal{O}(1)$, in practice we need roughly the same number of samples as MVD to obtain a low error gradient estimate. A more in-depth analysis of this
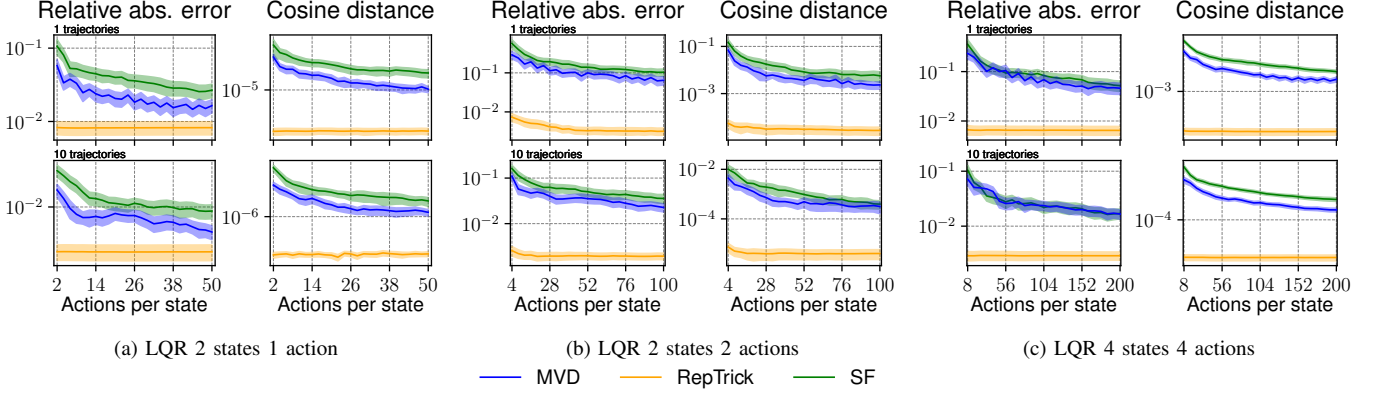
Fig. 2: Gradient errors in magnitude and direction in different LQRs, per number of trajectories and sampled actions. Depicted are the mean and the 95% confidence interval of 25 random seeds.
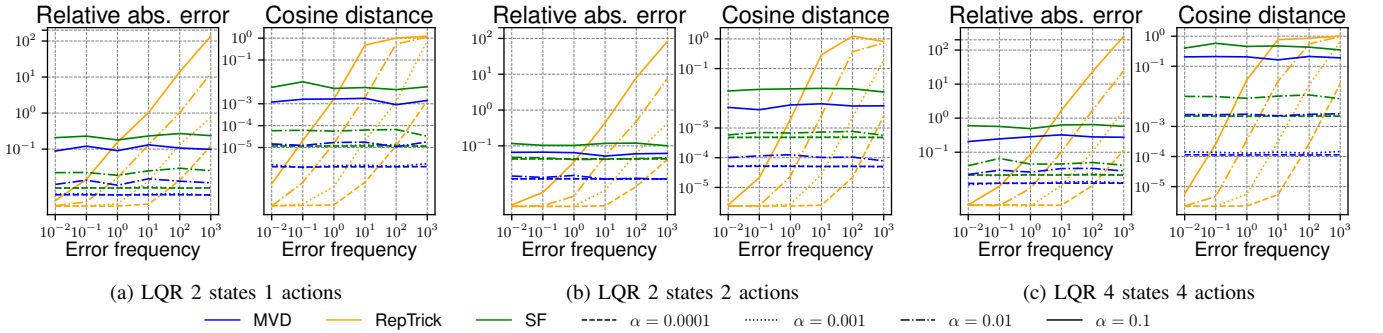

Fig. 3: Gradient errors in magnitude and direction in different LQRs, with increasing error noise and frequency. The different linestyles correspond to different error amplitudes α. The results are estimated using 10 trajectories and $20|\mathcal{A}|$ actions per state for each LQR. For MVD and SF the amplitudes 0.0001 and 0.001 show identical results and thus the lines are superposed. Depicted is the mean over 25 random seeds.
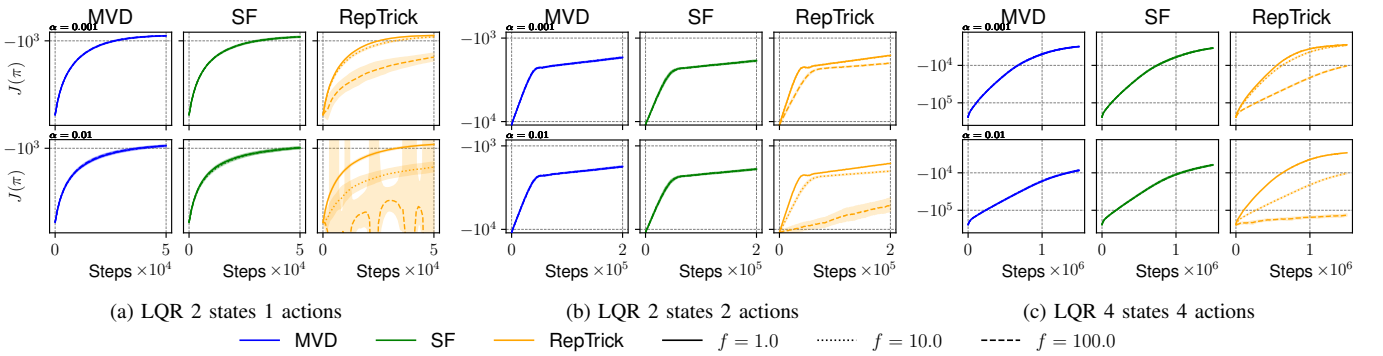

Fig. 4: Learning curves of expected return per collected transitions, for different error amplitudes and frequencies. The gradients are estimated using 1 trajectory and $2|\mathcal{A}|$ actions per state for each LQR. Depicted are the mean and the 95% confidence interval of 25 random seeds. The lines for different frequencies and the confidence intervals appear superposed in some plots.

fact is left for future work. Knowing that the Q-function is quadratic in the action, the results are in line with the example from Fig. 1a, where all estimators perform well.

Next, we consider a scenario where the true Q-function is not available and has to be estimated. Instead of fitting a state-action value function, we model the approximator with a local approximation error on top of the true value as $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha Q(\boldsymbol{s}, \boldsymbol{a}) \cos(2\pi f \boldsymbol{p}^{\intercal} \boldsymbol{a} + \phi)$. I.e., we use an additive sinusoidal error proportional to the true estimate, where α represents the fraction of the true Q amplitude, f is the error frequency, $\boldsymbol{p}$ is a random vector whose entries sum to 1, and $\phi \sim \mathcal{U}[0, 2\pi]$ a phase shift. $\boldsymbol{p}$ and ϕ introduce randomness to remove correlation between action dimensions. Fig. 3 depicts the gradient errors in magnitude and direction as a function of the error frequency and amplitude, which reveals two insights. The error frequency does not affect the gradient estimation of SF or MVD, which are only affected by the amplitude, as can be seen by the horizontal blue and green lines. On the contrary the Rep-trick is heavily affected by both variables. This result matches the theory, since under a Gaussian distribution the variance of the Rep-trick is upper-bounded by the Lipschitz constant of the derivative of Q [5]. Fig. 4 shows the expected discounted return per steps taken in the environment with different amplitudes and frequencies of errors. As expected, the SF and MVD do not suffer from errors in the estimation and converge towards the optimal policy in the presence of high-frequency error terms. The Rep-trick on the other hand either shows slower convergence or fails to converge due to the poor gradient estimates.

From these experiments we observe that even though the Rep-trick provides the most precise and accurate gradient under a true value function, this does not always hold, especially in the case where there is an action correlated error in the approximation.

VI. MVDs in Off-Policy Policy Gradient for Deep Reinforcement Learning

In the following, we illustrate how MVDs can be used in a deep RL algorithm, by using the off-policy actor-critic method SAC. The optimization objective is similar to the off-policy gradient [20] but with a soft Q-function, which includes an entropy regularization term. This additional term encourages exploration by preventing the policy from becoming too deterministic during learning. The surrogate objective optimized by SAC is

$$J_{\pi}(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim d^{\beta}, \boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s}; \boldsymbol{\omega})} \left[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}) - \alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\omega}) \right],$$

where d^{β} is an off-policy state distribution, the Q-function is a neural network parameterized by $\boldsymbol{\phi}$, α weighs the entropy regularization term, and $\boldsymbol{\omega}$ are the distributional parameters of π. For simplicity, we omit the parameterization of $\boldsymbol{\omega}$ as its extension is done as in (5).

SAC estimates the gradient w.r.t. $\boldsymbol{\omega}$ with the Rep-trick. Instead, we modify it by computing the gradient using MVD. Defining $f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega}) = Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}) - \alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\omega})$, the resulting off-policy policy gradient w.r.t. a parameter ω_k becomes

$$\nabla_{\omega_k} J_{\pi}(\boldsymbol{\omega}) = \mathbb{E}_{\boldsymbol{s} \sim d^{\beta}} \left[c_{\omega_k} \left(\mathbb{E}_{\boldsymbol{a} \sim \pi_{\omega_k}^{+}(\cdot|\boldsymbol{s}; \boldsymbol{\omega})} \left[f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega}) \right] \right. \right.$$
$$\left. - \mathbb{E}_{\boldsymbol{a} \sim \pi_{\omega_k}^{-}(\cdot|\boldsymbol{s}; \boldsymbol{\omega})} \left[f(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\phi}, \boldsymbol{\omega}) \right] \right)$$
$$\left. - \mathbb{E}_{\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s}; \boldsymbol{\omega})} \left[\nabla_{\boldsymbol{\omega}} \alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\omega}) \right] \right],$$

where $\pi_{\omega_k}^{+}$ and $\pi_{\omega_k}^{-}$ are the positive and negative components of the policy decomposition. We denote this modification as Soft Actor-Critic with MVD (SAC-MVD). Unlike in SAC, we do not require the Q-function to be differentiable, allowing the use of any type of function approximators.

Experimental Results

We benchmark the performance of SAC with the different gradient estimators in high-dimensional continuous control tasks, based on the PyBullet simulator [31]. We compare the original SAC with: SAC-MVD with one MC sample; SAC-SF - a version of SAC using the SF; SAC-SF-extra-samples - same as SAC-SF but with the same number of queries as SAC-MVD per gradient estimate; SAC-extra-samples - same as SAC but with the same number of gradient estimates as SAC-SF-extra-samples; DDPG [16] and TD3 [17] - two state-of-the-art off-policy algorithms, which are used as baseline comparisons. All variations of SAC use the same hyperparameters and neural network architectures for the value function and policy as in the original work [13].
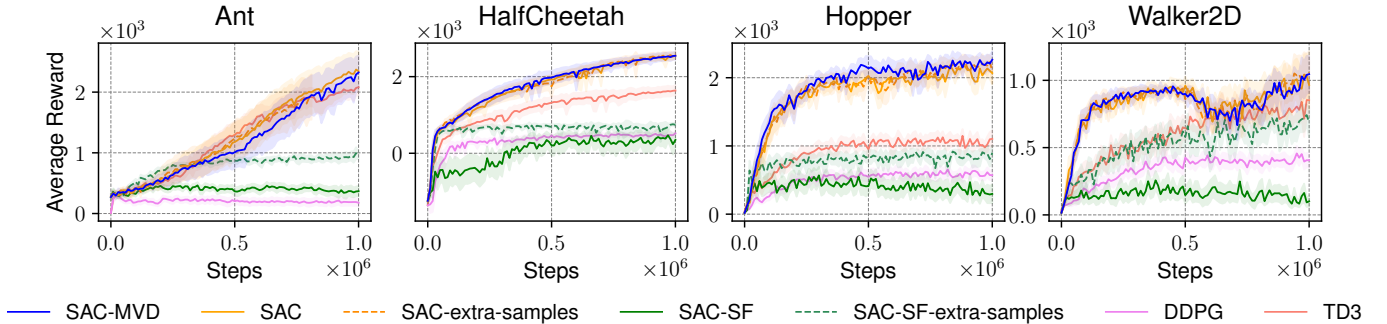

Fig. 5: Policy evaluation results during training on different tasks in deep RL. Depicted are the average reward per samples collected and the 95% confidence interval of 25 random seeds.

The average reward curves obtained during training are shown in Fig. 5. The original version of SAC and SAC-MVD show similar performance, and increasing the number of samples does not improve the results, as SAC-extra-samples performs equally well. This empirically shows that the MVD estimator and the Rep-trick are equally good to provide precise and accurate gradient estimates in these high-dimensional tasks. SAC-SF and SAC-SF-extra-samples fail to solve most of the tasks, since no baseline was used for variance reduction. This can be done with GAE or by estimating an extra value function. However, that can introduce more bias in the estimation. Therefore, we chose to not use a baseline for this experiment to compare directly all estimators in the base case.

The results of our experiments reveal that the Rep-trick is not fundamental to replicate the performance of SAC, and suggest that the superior performances of this algorithm depend on other aspects, such as the entropy regularization, the state-dependent covariance, and the squashed Gaussian policy. Additionally, it is worth remembering that MVDs are applicable to function classes where the Rep-trick is not, which allows to explore other function approximator classes and still use the benefits of SAC.

VII. MVDs in On-Policy Policy Gradient for Non-differentiable Approximators

In this section we present the results of the on-policy gradient with MVDs when using a non-differentiable Q-function approximator. The goal is to show that when the Rep-trick is not applicable, MVD-based methods can obtain comparable or better results than SF ones.

For the Q-function approximator we use Extra-Trees, a type of regression tree, due to their interpretability, low variance and bias properties, and better computational complexity compared to other tree methods [11]. Algorithm 1 shows a pseudo-code for the method we use – Tree-MVD-Policy Gradient (Tree-MVD). Fitting the Q-function in line 6 is done by applying Bellman Equation, $Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s', a'} [Q^\pi(s', a')]$, with bootstrapping for a fixed amount of iterations, where s, a and s' are samples collected on-policy and from a replay buffer, as is common in recent RL algorithms [32]. The latter helps in the generalization out of the current on-policy samples. The expectation over the next-action is solved with a single sample. This procedure is an extension of Expected SARSA [33].

Experimental Results

We evaluate this method in four environments with continuous state and action spaces. The first two tasks are the Pendulum-v0 and LunarLanderContinuous-v2 from OpenAI Gym [34], which are common baselines for continuous control. The remaining two are the Corridor and Room, which are depicted in Fig. 6. In both environments the agent starts at the green dot and the goal is to move towards the red area, where the episode terminates (or when a fixed amount of

---

Algorithm 1: Tree-MVD-Policy Gradient (Tree-MVD)

Input: Stochastic policy $\pi(a|\omega(s; \theta))$; E epochs; R replay buffer samples; P policy updates; B batch size policy update; N Monte Carlo samples; learning rate α

Result: Optimized policy parameters θ^*

1   $\mathcal{B} \leftarrow$ start a replay buffer
2   **for** $e = 0 \ldots E - 1$ **do**   // epochs
3      $\mathcal{D}_\pi \leftarrow$ collect data on-policy in the environment
4      $\mathcal{D}_\mathcal{B} \leftarrow$ get R samples from $\mathcal{B}$
5      $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{D}_\pi$ augment the replay buffer
6   Fit Q^π using Extra-Trees with $\mathcal{D}_\pi \cup \mathcal{D}_\mathcal{B}$
7   **for** $p = 0 \ldots P - 1$ **do**   // policy updates
8         $\mathcal{S} \leftarrow$ Sample B states from D_π
9   **for** $s \in \mathcal{S}$ **do**
10   **for** $k = 0 \ldots K - 1$ **do**   // dist params
11               $\hat{g}_k = c_{\omega_k} \frac{1}{N} \left(\sum_{i=1}^N Q^\pi(s, a_i^+) - Q^\pi(s, a_i^-) \right)$
12               $a_i^+ \sim \pi_{\omega_k}^+(\cdot|\omega(s; \theta)), \; a_i^- \sim \pi_{\omega_k}^-(\cdot|\omega(s; \theta))$
13   **end**
14            $\hat{g}_s^{\text{MVD}} = [\hat{g}_0, \ldots, \hat{g}_{K-1}]^\mathsf{T}$
15            $\hat{g}_s = \nabla_\theta \omega(s; \theta) \hat{g}_s^{\text{MVD}}$ // grad estimate
16   **end**
17         $\hat{g} = $ average $\left([\hat{g}_1, \ldots, \hat{g}_{|\mathcal{S}|}] \right)$
18         $\theta \leftarrow \theta + \alpha \hat{g}$
19   **end**
20   **end**

---

steps is reached). The state is the (x, y) position of the agent and the action the velocity vector, with bounded norm $\sqrt{2}$.
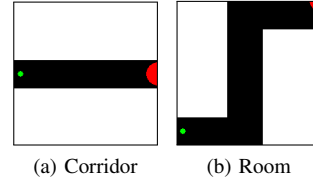

(a) Corridor   (b) Room

Fig. 6: Corridor and Room environments.

The reward function is the sum of a negative constant for being in the white area or hitting the wall, the negative euclidean distance between the current agent position and the goal, and a positive reward for reaching the red area. With these two environments we want to argue that due to the task structure, using a non-smooth approximator, such as a regression tree, is preferable to a smooth one, e.g. a neural network.

As a baseline comparison we choose the on-policy algorithms Proximal Policy Optimization (PPO) and TRPO, which use a neural network to approximate the value function and GAE to estimate the advantage function. For all tasks we use the same policy as in PPO [35] - a multivariate Gaussian distribution with diagonal covariance, where the mean is the output of a neural network and the log standard deviation a learnable state-independent parameter. The best performing hyperparameters for both methods were chosen with a grid search. Importantly, for Tree-MVD we use only one MC sample to solve the action expectations in the policy gradient.

Fig. 8 shows the average reward during training. The plots suggest that the MVD-based method performs comparable or better to the SF-based PPO. Fig. 7 presents the value functions learned by each algorithm at different training epochs in selected environments. In the Pendulum task (1st row), the value function learned with Tree-MVD at epoch 15 is close to the optimal one at epoch 60. This is in contrast to the one learned with PPO for the same epoch (2nd row), which

(a) Pendulum Tree-MVD

(b) Pendulum PPO

(c) Corridor Tree-MVD
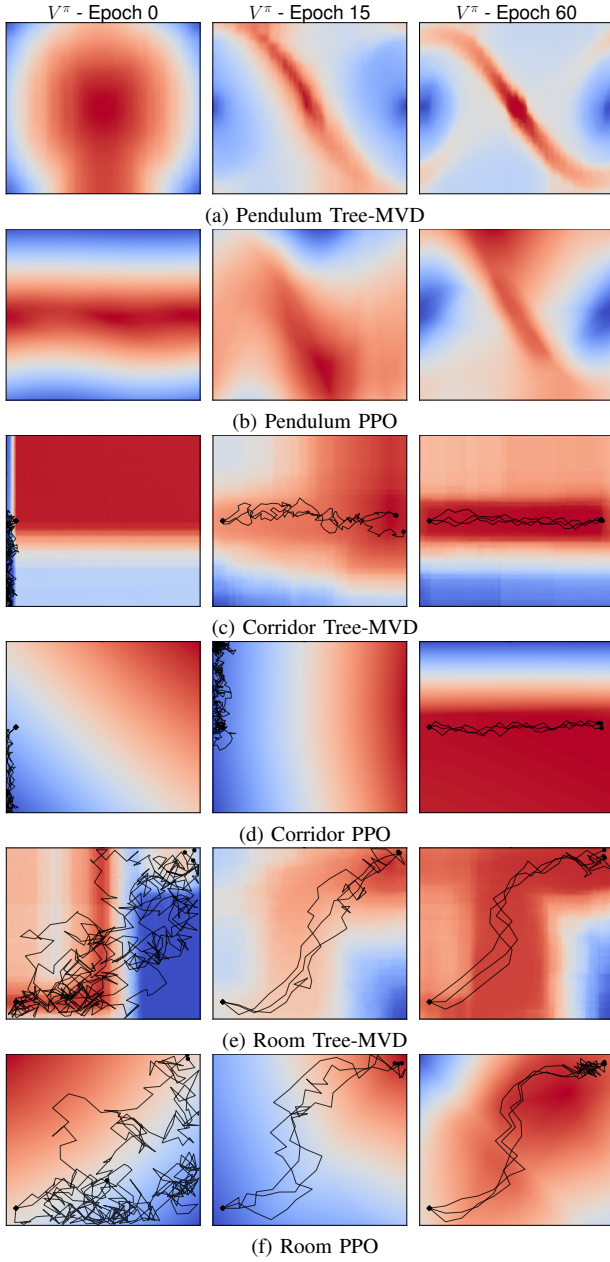
(d) Corridor PPO

(e) Room Tree-MVD

(f) Room PPO

Fig. 7: Value functions for Tree-MVD and PPO at the end of different training epochs. In the Corridor and Room tasks the lines depict agent trajectories during evaluation, where the diamond is the starting state and the circle the ending state. Coloring: lower values in blue, higher values in red.
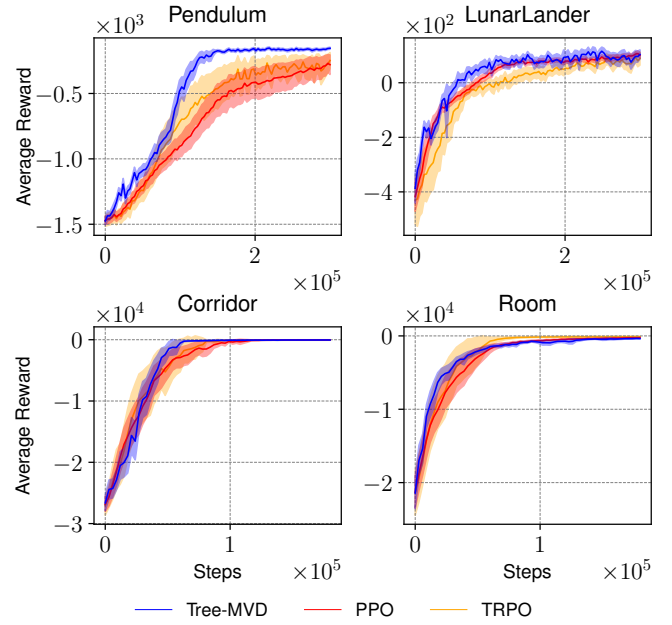

Fig. 8: Policy evaluation results during training on different tasks with on-policy gradient algorithms. Depicted are the average reward per samples collected and the 95% confidence interval of 25 random seeds.

the environments and reward descriptions from Fig. 6, and thus help accelerate learning. A version of Tree-MVD with SF showed unstable behaviors due to the absence of a value function estimator as baseline.

VIII. CONCLUSION

In this work we present MVDs as a complement to the SF and Rep-trick estimators for actor-critic policy gradient algorithms. We empirically show that methods based on this estimator are a viable alternative to the commonly used ones, and differently from [6], we avoid resetting the system to a specific state, which is impractical in real systems, showing how MVDs are applicable to the general RL framework.

Our experiments in step-based policy search highlight important facts about MVDs. In simple environments such as the LQR and with an oracle Q-function, the MVD performs better than the SF and worse than the Rep-trick. However, in the presence of a local error, the MVD and SF estimates are not affected by the error frequency, but only by the amplitude. On the contrary, the Rep-trick is sensitive to both, which suggests that it should not be used if the Q-function approximator changes abruptly. SF-based methods need to estimate an advantage function for variance reduction, while MVD only needs the Q-function approximation. We can use non-differentiable critics, in particular regression trees, which in certain environments can lead to faster convergence and compete with SF-based methods. The Rep-trick is not applicable to the latter case. MVDs can be used for high-dimensional action spaces in deep RL algorithms, and obtain comparable results with the Rep-trick using only one gradient

can explain the difference in performance in Fig. 8. In the Corridor and Room environments, we see the benefits of using regression trees instead of neural networks when the tasks exhibit a more structured representation. Compare the value functions in the Corridor task for both algorithms (3rd and 4th rows). Regression trees provide a better approximation of the optimal value function, are easier to interpret, and thus can partly explain the slight increase in performance over PPO in Fig. 8. The value functions found by Tree-MVD better match

estimate, which shows that the Rep-trick is not a crucial part of the SAC algorithm.

Which stochastic gradient estimator to use is problem dependent and an ongoing topic of research. The Rep-trick has shown in the recent years to achieve good results in VI, e.g. in Variational Autoencoders (VAEs) [9], and the SF in policy gradient algorithms such as PPO [35]. MVDs are rarely used in ML, however, they have been recently employed in approximate bayesian inference [27].

An interesting future research direction is to investigate how to reduce the computational complexity of MVDs e.g., by computing only the derivatives along certain parameter dimensions and using a convex combination of all estimators. This kind of estimator is still unbiased [5]. Moreover, a theoretical analysis of bias and variance for MVD is needed to improve our understanding of the estimators' properties and to determine in which kind of tasks one should be preferred over the others.

Given the empirical results presented in our work, we argue that the MVD estimator is a useful tool for developing novel algorithms to solve challenging control problems in Reinforcement Learning research.

REFERENCES

[1] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, 2004, pp. 2619–2624 Vol.3.

[2] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics." *Found. Trends Robotics*, vol. 2, no. 1-2, pp. 1–142, 2013.

[3] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, 1992.

[4] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 1999, pp. 1057–1063.

[5] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning," *Journal of Machine Learning Research*, vol. 21, no. 132, pp. 1–62, 2020.

[6] S. Bhatt, A. Koppel, and V. Krishnamurthy, "Policy gradient using weak derivatives for reinforcement learning," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 5531–5537.

[7] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Int. Res.*, vol. 15, no. 1, p. 319–350, Nov. 2001.

[8] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.

[9] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[10] E. Jang, S. Gu, and B. Poole, "Categorical reparametrization with gumbel-softmax," in *Proceedings International Conference on Learning Representations 2017*. OpenReviews.net, Apr. 2017.

[11] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, p. 3–42, Apr. 2006.

[12] P. Glasserman, *Monte Carlo methods in financial engineering*. New York: Springer, 2004.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of International Conference on Machine Learning (ICML)*, vol. 80, 2018, pp. 1856–1865.

[14] V. Krishnamurthy and F. V. Abad, "Real-time reinforcement learning of constrained markov decision processes with weak derivatives," 2011.

[15] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *International Conference on International Conference on Machine Learning (ICML)*, 2015, p. 1889–1897.

[16] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations, (ICLR)*, 2016.

[17] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 1587–1596.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*, 2014, p. 387–395.

[19] V. Krishnamurthy, K. Martin, and F. V. Abad, "Implementation of gradient estimation to a constrained markov decision problem," in *IEEE International Conference on Decision and Control*, vol. 5. IEEE, 2003, pp. 4841–4846.

[20] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Madison, WI, USA: Omnipress, 2012, p. 179–186.

[21] J. Schulman, N. Heess, T. Weber, and P. Abbeel, "Gradient estimation using stochastic computation graphs," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 2015, p. 3528–3536.

[22] P. W. Glynn, "Likelilood ratio gradient estimation: An overview," in *Proceedings of the 19th Conference on Winter Simulation*, 1987, p. 366–375.

[23] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomput.*, vol. 71, no. 7–9, p. 1180–1190, Mar. 2008.

[24] G. C. Pflug, "Sampling derivatives of probabilities," *Computing*, vol. 42, no. 4, pp. 315–328, 1989.

[25] B. Heidergott, G. Pflug, and F. J. Vazquez-Abad, "Measure-valued differentiation for stochastic systems: from simple distributions to markov chains," 2003. [Online]. Available: https://personal.vu.nl/b.f.heidergott/mvd.pdf

[26] B. Heidergott, F. Vázquez-Abad, and M. Gerad, "Measure valued differentiation for stochastic processes: The finite horizon case," *Technology Analysis and Strategic Management*, 2000.

[27] M. Rosca, M. Figurnov, S. Mohamed, and A. Mnih, "Measure-valued derivatives for approximate bayesian inference," in *4th workshop on Bayesian Deep Learning*, 2019.

[28] G. C. Pflug, *Optimization of Stochastic Models*. Springer, 1996.

[29] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 551–559, May 2010.

[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[31] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016, cite arxiv:1606.01540.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *CoRR*, vol. abs/1707.06347, 2017.

An Empirical Analysis of
Measure-Valued Derivatives for Policy Gradients

João Carvalho[1], Davide Tateo[1], Fabio Muratore[1,2], Jan Peters[1,3]
Intelligent Autonomous Systems
Technische Universität Darmstadt, Darmstadt, Germany
{joao,davide,fabio}@robot-learning.de, jan.peters@tu-darmstadt.de

Supplementary material

I. Code to replicate the experiments

The code is written in Python and makes use of PyTorch for automatic differentiation and the MushroomRL library for algorithm implementation and benchmarking https://github.com/MushroomRL/mushroom-rl. The code to reproduce the experiments is available at https://git.ias.informatik.tu-darmstadt.de/carvalho/mvd-rl.

II. Optimization Test Functions

In Fig. 1 we maximize $\mathbb{E}_{p(\boldsymbol{x};\boldsymbol{\omega})}\left[f(\boldsymbol{x})\right]$ via gradient ascent, where f are three 2-dimensional test functions and $p(\boldsymbol{x};\boldsymbol{\omega})$ is a multivariate Gaussian distribution with diagonal covariance $p(\boldsymbol{x};\boldsymbol{\omega}) = \mathcal{N}\left(\boldsymbol{x};\boldsymbol{\omega} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}\right)$, with $\boldsymbol{x} \in \mathbb{R}^2$ and $\boldsymbol{\omega} \in \mathbb{R}^4$. The covariance is parameterized by the logarithm of standard deviation. The learning rate is kept constant and equal to 5×10^{-4} for all functions and estimators. The MVD uses one gradient estimate between parameter updates, while the SF and Rep-trick use the mean of eight estimates, which is the number of queries to f done with MVD for one estimate. The SF uses the optimal baseline for black-box optimization as computed by the PGPE algorithm.

Quadratic function
$f : \mathbb{R}^2 \to \mathbb{R}$
$f(\boldsymbol{x}) = -\boldsymbol{x}^\mathsf{T}\boldsymbol{x}$
Starting distribution: $\mathcal{N}\left(\boldsymbol{x};\boldsymbol{\mu} = (-5,-5), \boldsymbol{\Sigma} = \mathrm{diag}\left(2^2, 2^2\right)\right)$

Himmelblau function
$f : \mathbb{R}^2 \to \mathbb{R}$
$f(x,y) = -(x^2 + y - 11)^2 - (x + y^2 - 7)^2$
Starting distribution: $\mathcal{N}\left(\boldsymbol{x};\boldsymbol{\mu} = (0,-6), \boldsymbol{\Sigma} = \mathrm{diag}\left(2^2, 2^2\right)\right)$

Styblinski function
$f : \mathbb{R}^2 \to \mathbb{R}$
$f(\boldsymbol{x}) = -\frac{1}{2}\sum_{i=1}^{2}\left(x_i^4 - 16x_i^2 + 5x_i\right)$
Starting distribution: $\mathcal{N}\left(\boldsymbol{x};\boldsymbol{\mu} = (0,0), \boldsymbol{\Sigma} = \mathrm{diag}\left(2^2, 2^2\right)\right)$

III. Linear-Quadratic Regulator

The discounted infinite-horizon discrete-time LQR problem is defined as

$$\arg\max_{\boldsymbol{s}_t,\boldsymbol{a}_t} J = \sum_{t=0}^{\infty} -\gamma^t \left(\boldsymbol{s}_t^\mathsf{T}\mathbf{Q}\boldsymbol{s}_t + \boldsymbol{a}_t^\mathsf{T}\mathbf{R}\boldsymbol{a}_t\right)$$
$$\text{s.t. } \boldsymbol{s}_{t+1} = \mathbf{A}\boldsymbol{s}_t + \mathbf{B}\boldsymbol{a}_t.$$

The optimal control policy for this problem is a linear-in-the-state time-independent feedback controller $\boldsymbol{a}_t = -\mathbf{K}_{\text{opt}}\boldsymbol{s}_t$. In the experiments we compute the gradient w.r.t. $\mathbf{K}$ of a stochastic policy $\boldsymbol{a} \sim \mathcal{N}\left(\cdot\,|-\mathbf{K}\boldsymbol{s}_t, \boldsymbol{\Sigma}\right)$, with fixed diagonal covariance 0.1^2 in all action dimensions.

We build four environments with different dimensions of states and actions with $(|\mathcal{S}|, |\mathcal{A}|)$: $(2,1)$, $(2,2)$, $(4,4)$ and $(6,6)$. The matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}$ and $\mathbf{R}$ can be found in the accompanying code. The dynamics matrices $\mathbf{A}$ and $\mathbf{B}$ are non-diagonal, which makes the LQR more difficult to solve. The matrix $\mathbf{A}$ is chosen such that the system is unstable when $\boldsymbol{a}_t = 0$. The initial gain matrix $\mathbf{K}_{\mathrm{init}}$ is chosen by sampling from a Gaussian distribution with mean $\mathbf{K}_{\mathrm{opt}}$ and problem-dependent covariance, such that the closed-loop system is stable. The initial state for each environment is 9 for each dimension. For instance, in LQRs $(2,1)$ and $(2,2)$ the initial state is $\boldsymbol{s}_0 = (9,9)^\mathsf{T}$. The discount factor is $\gamma = 0.99$. To simulate infinite horizons, trajectory rollouts have $T = 1000$ steps. The policy gradients are computed from $\boldsymbol{s}_0$ and $\mathbf{K}_{\mathrm{init}}$. The discounted state distribution is obtained by sampling trajectories and multiplying each state at time t with γ^t.

The experiments of Fig. 2 use the true Q and V functions computed with $\mathbf{K}_{\mathrm{init}}$. Figures 3 and 4 simulate an error in the Q-function estimator with an added local sinusoidal term to the true state-action value function. This approximator is modelled as $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha Q(\boldsymbol{s}, \boldsymbol{a}) \cos(2\pi f \boldsymbol{p}^\mathsf{T} \boldsymbol{a} + \phi)$. f is the error frequency, $\boldsymbol{p}$ is a random vector whose entries sum to 1, and $\phi \sim \mathcal{U}[0, 2\pi]$ a phase shift. $\boldsymbol{p}$ and ϕ introduce randomness to remove correlation between action dimensions. The factor α represents an error proportional to the true value, and the cosine term models adding a (high) frequency error component. These frequency components can appear in function approximators, especially if they overfit to the data. This is a simplified error model that uses only one frequency component, but it is useful to understand the sensitivity of the gradient estimators to local errors in function approximators.

Figures 9 and 10 complement the results of Fig. 2 with more trajectories, and a 6-dimensional LQR.

In Fig. 4 the initial policy is the same as in Fig. 2. The policy update uses the Adam optimizer and the following learning rates $(|\mathcal{S}|, |\mathcal{A}|)$: $(2,1) : 5 \times 10^{-2}$; $(2,2) : 1 \times 10^{-2}$; $(4,4) : 3 \times 10^{-3}$; $(6,6) : 5 \times 10^{-3}$. Fig. 11 complements the results without added noise ($\alpha = 0$), and a 6-dimensional LQR.
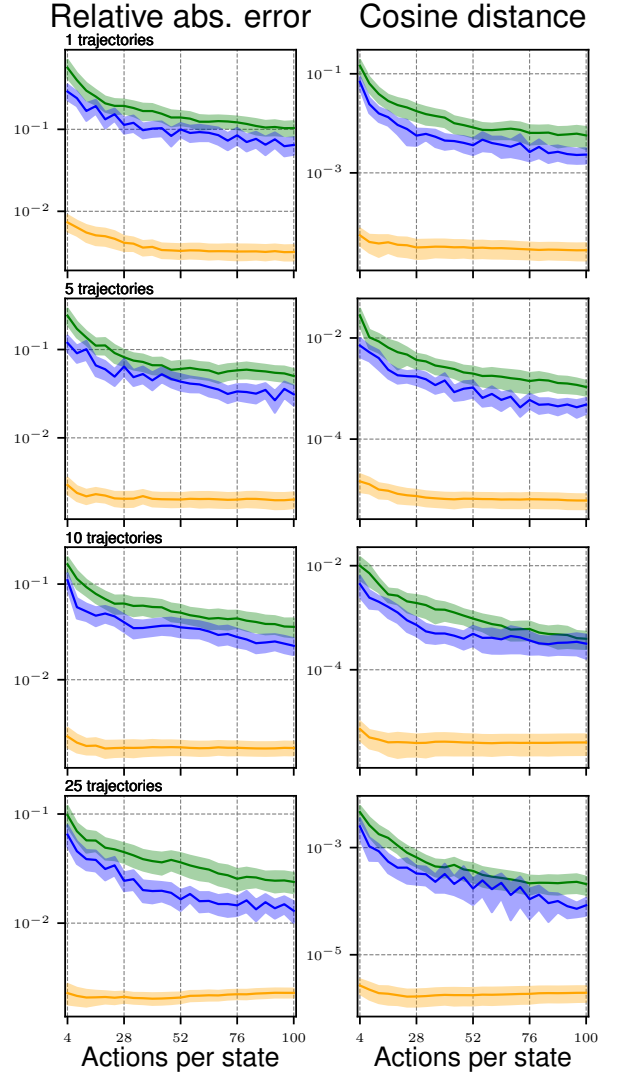
Fig. 9: Gradient errors in magnitude and direction in the LQRs (2 states, 1 action) and (2 states, 2 actions), per number of trajectories and sampled actions. Depicted are the mean and the 95% confidence interval of 25 random seeds.
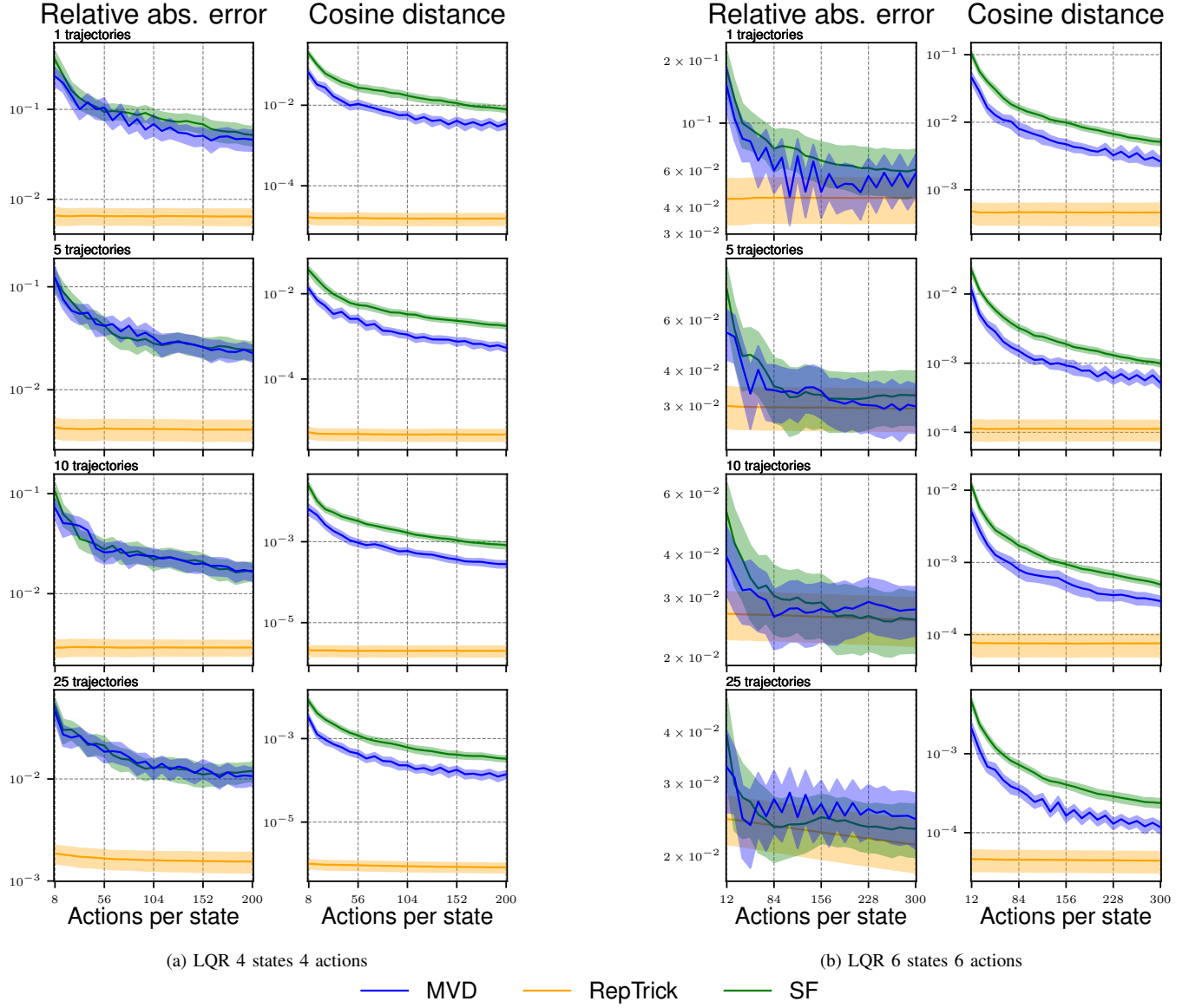
Fig. 10: Gradient errors in magnitude and direction in the LQRs (4 states, 4 actions) and (6 states, 6 actions), per number of trajectories and sampled actions. Depicted are the mean and the 95% confidence interval of 25 random seeds.

(a) LQR 2 states 1 actions

(b) LQR 2 states 2 actions

(c) LQR 4 states 4 actions
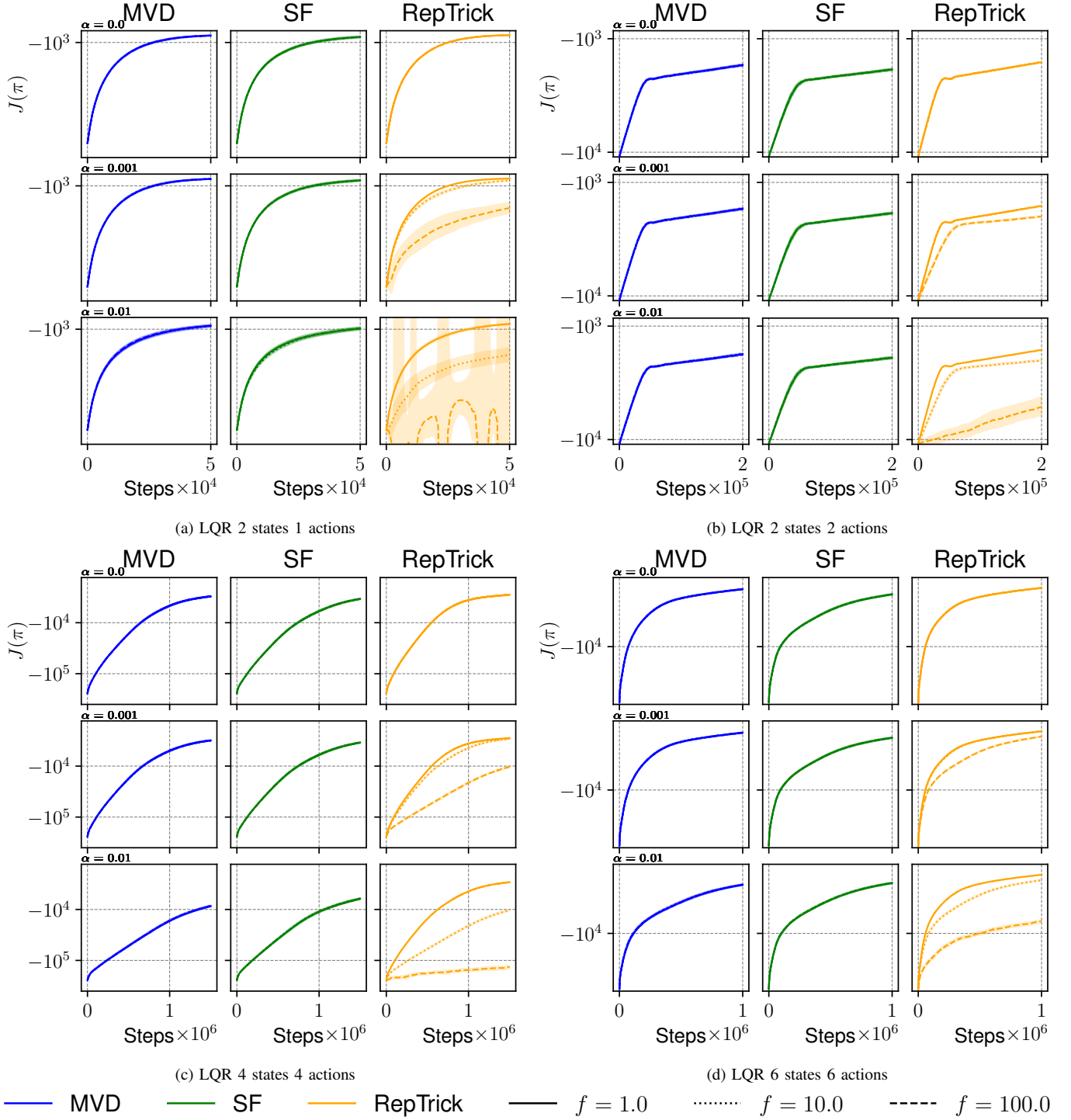
(d) LQR 6 states 6 actions

Fig. 11: Learning curves of the LQR tasks with an error in the Q-function approximator. Noise amplitudes (0.001, 0.01, 0.1, 1.0, 10.0), bottom to top.

IV. OFF-POLICY EXPERIMENTS

Off-policy experiments from Fig. 5 use the environments from the PyBullet simulator [31]. Fig. 12 shows additional results in simpler tasks. Table III contain the used hyperparameters. The neural network architectures are from the original papers.
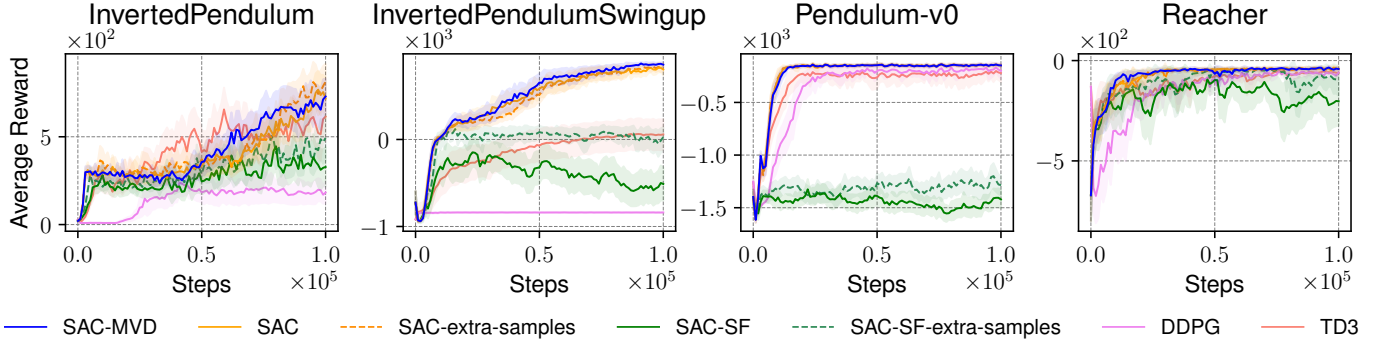

Fig. 12: Policy evaluation results per samples collected during training on different tasks in deep RL. Depicted are the average reward and the 95% confidence interval of 25 random seeds.

	Pendulum-v0	InvertedPendulum-v0 InvertedPendulumSwingup-v0 ReacherEnv-v0	Ant-v0, HalfCheetah-v0 Walker2d-v0, Hopper-v0
SAC variants			
horizon	200	1000	1000
γ	0.99	0.99	0.99
epochs	50	50	100
steps/epoch	1000	1000	10000
episodes evaluation	10	10	10
batch size	64	64	256
warmup transitions	128	128	10000
max replay size	500000	500000	500000
critic network	[64, 64] ReLU	[64, 64] ReLU	[256, 256] ReLU
actor network	[64, 64] ReLU	[64, 64] ReLU	[256, 256] ReLU
optimizer	Adam	Adam	Adam
lr actor	1×10^{-4}	1×10^{-4}	1×10^{-4}
lr critic	3×10^{-4}	3×10^{-4}	3×10^{-4}
DDPG and TD3			
batch size	64	64	256
warmup transitions	128	128	10000
max replay size	1000000	1000000	1000000
critic network	[64, 64] ReLU	[64, 64] ReLU	[400, 300] ReLU
actor network	[64, 64] ReLU	[64, 64] ReLU	[400, 300] ReLU
optimizer	Adam	Adam	Adam
lr actor	1×10^{-4}	1×10^{-4}	1×10^{-4}
lr critic	1×10^{-3}	1×10^{-3}	1×10^{-3}

TABLE III: Hyperparameters for the off-policy experiments.

V. On-Policy Experiments

For TRPO and PPO the policy is a Gaussian distribution with diagonal covariance, where the mean is the output of a neural network, and the log-standard deviation is a state independent learnable parameter, as in the original papers. Tree-MVD optimizes the same policy but applies a `tanh` operator to the sampled actions, as done in SAC. Applying this operator to MVDs is straight forward.

Tables IV and V contain the hyperparameters used in the experiments. The neural network architectures are taken from the original works.

	Pendulum-v0	LunarLanderContinuous-v2	Room	Corridor
horizon	200	1000	300	300
γ	0.99	0.99	0.99	
epochs	100	100	60	60
steps/epoch	3000	3000	2000	2000
episodes evaluation	10	10	10	10
iters bellman equation	100	50	25	25
tree estimators	100	50	25	25
min samples split node	2	2	8	8
min samples leaf node	1	1	4	4
max replay size	500000	500000	500000	500000
replay batch size	25000	25000	10000	10000
actor update epochs	4	4	4	4
actor batch size	256	128	128	128
actor network	[32, 32] ReLU	[32, 32] ReLU	[32, 32] ReLU	[32, 32] ReLU
optimizer	Adam	Adam	Adam	Adam
actor learning rate	3×10^{-4}	3×10^{-4}	1×10^{-4}	1×10^{-4}
initial σ	1	1	1	1

TABLE IV: Hyperparameters for the on-policy experiments with Tree-MVD.

	Pendulum-v0	LunarLanderContinuous-v2	Room	Corridor
horizon	200	1000	300	300
γ	0.99	0.99	0.99	
epochs	100	100	60	60
steps/epoch	3000	3000	2000	2000
episodes evaluation	10	10	10	10
critic update epochs	10	10	10	10
critic batch size	64	64	64	64
critic network	[32, 32] ReLU	[32, 32] ReLU	[128, 128] ReLU	[128, 128] ReLU
critic learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
actor update epochs	8	4	4	4
actor batch size	256	256	128	128
actor network	[32, 32] ReLU	[32, 32] ReLU	[32, 32] ReLU	[32, 32] ReLU
optimizer	Adam	Adam	Adam	Adam
actor learning rate	3×10^{-4}	3×10^{-4}	1×10^{-4}	1×10^{-4}
initial σ	1	1	1	1
PPO	$\epsilon = 0.2$, $\lambda(\text{GAE}) = 0.95$			
TRPO	$\max \text{KL} = 0.01$, $\lambda(\text{GAE}) = 0.95$, epochs line search $= 10$, epochs conj gradient $= 100$			

TABLE V: Hyperparameters for the on-policy experiments with PPO and TRPO.