# Model-free Policy Learning with Reward Gradients

**Qingfeng Lan**
University of Alberta

**Samuele Tosatto**
University of Alberta

**Homayoon Farrahi**
University of Alberta

**A. Rupam Mahmood**
University of Alberta
CIFAR AI Chair, Amii

## Abstract

Despite the increasing popularity of policy gradient methods, they are yet to be widely utilized in sample-scarce applications, such as robotics. The sample efficiency could be improved by making best usage of available information. As a key component in reinforcement learning, the reward function is usually devised carefully to guide the agent. Hence, the reward function is usually known, allowing access to not only scalar reward signals but also reward gradients. To benefit from reward gradients, previous works require the knowledge of environment dynamics, which are hard to obtain. In this work, we develop the *Reward Policy Gradient* estimator, a novel approach that integrates reward gradients without learning a model. Bypassing the model dynamics allows our estimator to achieve a better bias-variance trade-off, which results in a higher sample efficiency, as shown in the empirical analysis. Our method also boosts the performance of Proximal Policy Optimization on different MuJoCo control tasks. [1]

# 1 INTRODUCTION

Policy gradient methods are increasingly popular in the reinforcement learning (RL) community due to their robustness and ability to deal with continuous action spaces. They have been employed in a large variety of domains, including computer games (Vinyals et al., 2019; Berner et al., 2019; Zha et al., 2021; Badia et al., 2020), simulated robotic tasks (Haarnoja et al., 2018; Lillicrap et al., 2016), and recommendation systems (Zheng et al., 2018; Zhao et al., 2018; Afsar et al., 2021). Despite their popularity, policy gradient methods still suffer from low sample efficiency, which hinders their applicability to complex real-world situations. One way to mitigate this inefficiency, is to make best use of prior knowledge into the learning system.

In most real-world applications, reward functions are manually designed (Dulac-Arnold et al., 2019), and therefore known in advance. Most of state-of-the-art techniques, however, do not benefit from such prior knowledge. One way to take full advantage of the reward function is to use its gradient w.r.t. the policy parameters. Attempts to use reward gradients are already observed in prior works, such as Deterministic Value-Policy Gradient (DVPG) algorithm (Cai et al., 2020), Stochastic Value Gradient (SVG) methods (Heess et al., 2015), and Dreamer (Hafner et al., 2019). However, all these methods require a learned transition model, which is well-known to be a harder task than learning value functions; an inaccurate model may even bring negative effect into training (van Hasselt et al., 2019; Jafferjee et al., 2020). In this work, we ask this question: *how can we benefit from reward gradients without learning a model?*

To answer this question, we develop a new policy gradient estimator—the *Reward Policy Gradient* (RPG) estimator—that incorporates reward gradients. Specifically, in order to apply reward gradients while avoiding the use of a transition model, our approach incorporates two distinct gradient estimators that are popularly used in model-free learning — the likelihood ratio (LR) estimator and the reparameterization (RP) estimator. Broadly speaking, the LR estimator is generally applicable to both discrete and continuous variables, but often suffers from high variance and thus is frequently used in conjunction with variance reduction techniques (Williams, 1992; Schulman et al., 2017). The LR estimator is also a key component in the pol-

[1] The code is available at `https://github.com/qlan3/Explorer/tree/RPG`.

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. 

icy gradient theorem (Sutton et al., 2000). On the other hand, the RP estimator exhibits lower variance in practice (Greensmith et al., 2004), and its recent utilization in policy gradient methods that improved sample efficiency in several benchmark tasks (Haarnoja et al., 2018; Lillicrap et al., 2016; Fujimoto et al., 2018).

Our RPG estimator combines LR and RP gradients to avoid the need for a functional approximation of the model, while taking full advantage of reward gradients. Based on this new estimator, we propose a new on-policy policy gradient algorithm—the RPG algorithm—that utilizes and takes advantage of both LR and RP gradients. We empirically show that by incorporating reward gradients into the gradient estimator, the bias and variance of estimated gradient decrease significantly. To analyze the benefit of reward gradients and the properties of our estimator, we test our algorithm on bandit and simple Markov decision processes, where the ground truth gradient is known in closed-form. Moreover, we compare RPG with a state-of-the-art actor-critic algorithm—Proximal Policy Optimization—on challenging problems, showing that our algorithm outperforms the baseline algorithm.

In the rest of the paper, we first introduce some background knowledge for the LR and RP estimators. We then move to our major theoretical result — the Reward Policy Gradient theorem. Then we present RPG algorithm as well as the experimental results.

# 2 BACKGROUND

Consider a Markov decision process (MDP), $M = (\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma)$, where $\mathcal{S}$ is the continuous state space, $\mathcal{A}$ is the continuous action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ is the state transition probability distribution, $p_0 : \mathcal{S} \to [0, \infty)$ is the initial state probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,[2] and $\gamma \in [0, 1)$ is the discount factor. In a given MDP $M$, an agent interacts with the environment to generate a trajectory based on a policy distribution $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty)$. Specifically, the agent starts at some state $S_0 \sim p_0(\cdot)$; at each time-step $t = 0, 1, 2, \ldots$, the agent samples an action $A_t \in \mathcal{A}$ according to the policy $\pi$ (i.e. $A_t \sim \pi(\cdot|S_t)$), receives the immediate reward $R_t = r(S_t, A_t)$, and observes the next state $S_{t+1}$ sampled from the transition probability distribution (i.e. $S_{t+1} \sim p(\cdot|S_t, A_t)$). A trajectory up to time-step $T$ is defined as $\tau_T = (S_0, A_0, R_1, S_1, \cdots, S_T)$. We define the total discounted reward from time-step $t$ over $\tau_T$ as return $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r(S_k, A_k)$. Value functions are defined to be the expected return under policy $\pi$, $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$; similarly, action-value functions are defined as $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$.

Furthermore, the state-values and action-values can be related as $q_\pi(s, a) = r(s, a) + \gamma \int p(s'|s, a) v_\pi(s') \mathrm{d}s'$ and $v_\pi(s) = \int \pi(a|s) q_\pi(s, a) \mathrm{d}a$. The goal of the agent is to obtain a policy $\pi$ that maximizes the expected return starting from initial states. Let a policy $\pi_\theta$ be a differentiable function of a weight vector $\theta$. Our goal is to find $\theta$ that maximizes

$$J(\theta) = \int p_0(s) v_{\pi_\theta}(s) \mathrm{d}s. \tag{1}$$

To this end, we can apply gradient ascent techniques. Since the true gradient $\nabla_\theta J(\theta)$ is not typically available, we resort to Monte Carlo methods (Mohamed et al., 2020). This gradient estimation problem can be formalized as computing the unbiased gradient of an expectation of a function with respect to some parameters of a distribution. Specifically, let $p_\theta(x)$ be the probability distribution of $x$ with parameters $\theta$. Define $F(\theta) = \int p_\theta(x) \phi(x) \mathrm{d}x$. Then the key step of the problem can be formally defined as estimating $\nabla_\theta F(\theta)$:

$$\nabla_\theta F(\theta) = \nabla_\theta \mathbb{E}_{X \sim p_\theta}[\phi(X)]. \tag{2}$$

The gradient estimation problem is fundamental in many machine learning areas, such as reinforcement learning (Williams, 1992), variational inference (Hoffman et al., 2013), evolutionary algorithms (Conti et al., 2018), and variational auto-encoders (Kingma and Welling, 2013). In general, there are many approaches for gradient estimation, such as likelihood-ratio gradient estimators, reparameterization gradient estimators, finite difference methods, and mean-valued derivative methods (Pflug, 1989; Carvalho et al., 2021). Next, we introduce two major approaches for solving this problem.

### Likelihood-Ratio Gradient Estimators

The likelihood-ratio (LR) gradient estimator (Glynn, 1990) is one of the most popular gradient estimators. This estimator applies the log-derivative technique $\nabla_\theta \log p_\theta(x) = \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)}$ to obtain the unbiased gradient estimations:

$$\begin{aligned}
&\nabla_\theta F(\theta) = \nabla_\theta \mathbb{E}_{p_\theta(x)}[\phi(x)] = \nabla_\theta \int p_\theta(x) \phi(x) \mathrm{d}x \\
&= \int \phi(x) \nabla_\theta p_\theta(x) \mathrm{d}x = \int \phi(x) p_\theta(x) \nabla_\theta \log p_\theta(x) \mathrm{d}x \\
&= \mathbb{E}_{X \sim p_\theta}[\phi(X) \nabla_\theta \log p_\theta(X)].
\end{aligned} \tag{3}$$

The LR estimator is a fundamental component of the policy gradient theorem (Sutton and Barto, 2018).

[2] For simplicity, we assume $r$ is a deterministic function. In the stochastic case, $r$ can be defined as the expected reward for a given state-action pair.

Partly because of its generality of being applicable to both continuous and discrete action spaces, the LR estimator has been used in many policy gradient methods in RL. Many actor-critic algorithms also use this estimator to estimate gradient, such as Asynchronous Advantage Actor-Critic (Mnih et al., 2016), Trust Region Policy Optimization (Schulman et al., 2015a), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Actor-Critic with Experience Replay (Wang et al., 2017).

In practice, variance reduction (Greensmith et al., 2004) is usually necessary to fully exert the power of this estimator. Subtracting a baseline (Williams, 1992) from $\phi(x)$, applying eligibility traces (Singh and Sutton, 1996), and utilizing the Generalized Advantage Estimator (GAE) (Schulman et al., 2015b) are three effective approaches to mitigate the variance issue of policy gradient based on the LR estimator.

**Reparameterization Gradient Estimators**

The reparameterization (RP) gradient estimator is also known as the pathwise gradient estimator or the reparameterization technique (Mohamed et al., 2020). Given the underlying probability distribution $p_\theta(x)$, this estimator takes the advantage of the knowledge of distribution $p_\theta(x)$ and reparameterize $p_\theta(x)$ with a simpler base distribution $p(\epsilon)$ that makes two equivalent sampling processes:

$$X \sim p_\theta(\cdot) \iff X = f_\theta(\epsilon),\ \epsilon \sim p(\cdot), \tag{4}$$

where $f_\theta$ is a function that maps $\epsilon$ to $x$. In other words, there are two equivalent ways to sample $X \sim p_\theta(x)$: one is to sample it directly; the other way is to first sample $\epsilon$ from a base distribution $p(\epsilon)$ and then apply a function $f$ to transform $\epsilon$ to $X$. For example, assume $X$ is a Gaussian variable where $X \sim \mathcal{N}(\mu, \sigma)$ and $\theta = [\mu, \sigma]$. Let the base distribution be $p(\epsilon) = \mathcal{N}(0, 1)$. Then $X$ can be reparameterized as $X = f_\theta(\epsilon) = \mu + \sigma\epsilon$. For many common continuous distributions (e.g., the Gaussian, Log-Normal, Exponential, and Laplace), there exists such ways to reparameterize them with a simpler base distribution. Finally, we can write the gradient estimation as:

$$\begin{aligned}\nabla_\theta F(\theta) &= \nabla_\theta \int p_\theta(x)\phi(x)\mathrm{d}x \\ &= \nabla_\theta \int p(\epsilon)\phi(f_\theta(\epsilon))\mathrm{d}\epsilon = \int p(\epsilon)\nabla_\theta\phi(f_\theta(\epsilon))\mathrm{d}\epsilon. \end{aligned} \tag{5}$$

Note that $p_\theta(x) = p(\epsilon)|\nabla_\epsilon f_\theta(\epsilon)|^{-1}$ due to integration by substitution. RP estimators are only applicable to known and continuous distributions.[3] In general, there is no guarantee that RP outperforms LR (Gal, 2016; Parmas et al., 2018). However, RP estimators have a lower variance under certain assumptions (Xu et al., 2019), which usually leads to great benefits in many areas (Mohamed et al., 2020). Kingma and Welling (2013) applied the RP estimator to obtain a differentiable estimator of the variational lower bound that can be optimized directly using standard stochastic gradient methods. Rezende et al. (2014) used RP estimators for deep generative models and proposed Deep Latent Gaussian Models that are able to generate realistic images.

In many RL algorithms, RP estimators play a crucial role to reparameterize actions and decouple the randomness from a policy, such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018), SVG (Heess et al., 2015), and RELAX (Grathwohl et al., 2018). Deterministic Policy Gradient algorithm (DPG) (Silver et al., 2014), and Deep Deterministic Policy Gradient algorithm (DDPG) (Lillicrap et al., 2016). Deterministic Value-Policy Gradient algorithm (Cai et al., 2020) can also be viewed as special cases of these algorithms, where the probability density function of the base distribution is a Dirac delta function. Finally, Wang et al. (2019) proposed a class of RL algorithms called Reparameterizable RL, where the randomness of the environment is decoupled from the trajectory distribution via the reparameterization technique.

# 3 REWARD POLICY GRADIENT THEOREM

In this section, we first present our main theoretical result—the *Reward Policy Gradient Theorem*—a new policy gradient theorem that incorporates the gradient of the reward function without using a state transition function explicitly. The reward policy gradient theorem requires perfect knowledge of the reward and the value function. We show that an unbiased estimate can be obtained using approximated reward and state-value functions, by defining a set of compatible features, similarly to Sutton et al. (2000).

We begin by assuming that the action $A$ is sampled from the policy $\pi$ parameterized with $\theta$ given the current state $S$: $A \sim \pi_\theta(\cdot|S)$. We reparameterize the policy with a function $f$, $A = f_\theta(\epsilon; S)$, $\epsilon \sim p(\cdot)$. Let function $g$ be the inverse function of $f$, that is, $\epsilon = g_\theta(A; S)$ and $A = f_\theta(g_\theta(A; S); S)$. Furthermore, following Imani et al. (2018), we make two common assumptions on the MDP.

[3] An application to discrete random variables is possible by reparameterizing the Gumble distribution—the continuous counterpart of the categorical distribution (Jang et al., 2017).

**Assumption 1.** *$\mathcal{S}$ and $\mathcal{A}$ are closed and bounded.*

**Assumption 2.** *$p(s'|s,a)$, $f_\theta(\epsilon;s)$, $g_\theta(a|s)$, $\pi_\theta(a|s)$, $p(\epsilon)$, $r(s,a)$, $p_0(s)$ and their derivatives are continuous in all variables $s$, $a$, $s'$, $\theta$, and $\epsilon$.*

**Remark 1.** *The two assumptions above allow us to exchange derivatives and integrals, and the order of multiple integrations, using Fubini's theorem and Leibniz integral rule.*

We use the same objective $J(\theta)$ as in Eq. 1. Then under these two assumptions, we have the following results. The details of all proofs can be found in the appendix.

**Theorem 1** (Reward Policy Gradient)**.** *Suppose that the MDP satisfies Assumption 1 and 2, then*

$$\begin{aligned}&\nabla_\theta J(\theta)\\&= \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\big[\nabla_\theta r(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)}\\&+\gamma v_{\pi_\theta}(s')\nabla_\theta \log \pi_\theta(a|s)\big]\,\mathrm{d}s\mathrm{d}a\mathrm{d}s',\end{aligned}$$

*where $d^{\pi_\theta}(s') = \int \sum_{t=0}^{\infty} \gamma^t p_0(s) p(s \to s', t, \pi_\theta)\mathrm{d}s$ is the (discounted) stationary state distribution for policy $\pi_\theta$ and $p(s \to s', t, \pi_\theta)$ is the transition probability from $s$ to $s'$ with $t$ steps under policy $\pi_\theta$.*

*Proof sketch of Theorem 1.* We first apply the policy gradient theorem (Sutton et al., 2000) and get an intermediate result in form of the action-value function. Next, we split the action-value function in the policy gradient theorem into two parts: the immediate reward and the state-value of the next state. To incorporate reward gradients, we use the RP technique to the immediate reward part; to avoid the knowledge of the model, we apply the LR estimator to the state-value part. Finally, we combine both parts into an unbiased gradient estimation.

This theorem provides a new way of computing the objective gradients. Specifically, it presents the objective gradient in terms of both LR and RP gradients as additive components. The theorem also presents the first model-free unbiased gradient estimator of the objective function that utilizes gradients of the reward function. Some algorithms also estimate the gradient of the objective using gradients of the reward function, such as DVPG (Cai et al., 2020) and SVG (Heess et al., 2015). However, they are model-based algorithms in the sense that they require the knowledge of the state transition function to estimate an unbiased gradient, while our theorem points out a model-free approach without the knowledge of the state transition.

**Theorem 2** (Reward Policy Gradient with Function Approximation)**.** *Consider a parametric approximation of the reward function $\hat{r}_\omega(s,a)$ and a parametric approximation of the value function $\hat{v}_\phi(s)$ such that*

$$\begin{aligned}&\nabla_a \hat{r}_\omega(s,a) = \nabla_\theta^\intercal f_\theta(\epsilon;s)|_{\epsilon=g_\theta(a;s)}\omega,\\&\int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\nabla_\phi \hat{v}_\phi(s')\mathrm{d}s\mathrm{d}a\\&= \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\nabla_\theta \log \pi_\theta(a|s)\mathrm{d}s\mathrm{d}a,\end{aligned}$$

*where $\phi = \arg\min_\phi \mathbb{E}\left[(\hat{v}_\phi(s') - v(s'))^2\right]$ and*

$$\omega = \arg\min_\omega \sum_i \mathbb{E}\left[\left(\frac{\partial}{\partial\theta_i}\hat{r}_\omega(s,a) - \frac{\partial}{\partial\theta_i} r(s,a)\right)^2\right].$$

*Then,*

$$\begin{aligned}\nabla_\theta J(\theta) = \mathbb{E}\big[&\nabla_\theta \hat{r}_\omega(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)}\\&+\gamma\hat{v}_\phi(s')\nabla_\theta \log \pi_\theta(a|s)\big].\end{aligned}$$

*Note that all expectations are computed under $s \sim d^{\pi_\theta}$, $a \sim \pi_\theta(\cdot|s)$, $s' \sim p(\cdot|s,a)$.*

By this theorem, we show that when the reward function and the state-value function are approximated by sufficiently good function approximators (e.g., neural networks), we can obtain an unbiased gradient estimation under certain assumptions. The functions $\hat{r}_\omega$ and $\hat{v}_\phi$ are also known as compatible approximators (Sutton et al., 2000; Peters and Schaal, 2008). Note that $\phi$, $\omega$, and $\theta$ have same dimensions.

# 4 A REWARD POLICY GRADIENT ALGORITHM BASED ON PPO

Based on the reward policy gradient theorem, many different policy gradient algorithms can be developed that benefit from reward gradients without using a transition model. In this section, we develop a new policy gradient algorithm based on an existing deep policy gradient method called Proximal Policy Optimization (PPO). First, we introduce the baseline subtraction technique which is generally used in policy gradient algorithms to reduce variance. To be specific, we subtract the baseline $v_{\pi_\theta}(s)$ from $\gamma v_{\pi_\theta}(s')$ in the RPG estimator:

$$\nabla_\theta r(s, f_\theta(\epsilon;s)) + (\gamma v_{\pi_\theta}(s') - v_{\pi_\theta}(s))\nabla_\theta \log \pi_\theta(a|s). \tag{6}$$

Note that no bias is introduced in this step (Sutton and Barto, 2018).

In practice, when the state-value function and the reward function are unknown to the agent, we could use function approximators (e.g., neural networks) to

**Algorithm 1** RPG

Input: initial policy parameters $\theta$, value estimate parameters $\phi$, and reward estimate parameters $w$.
**for** $k = 1, 2, \ldots$ **do**
  Collect trajectories $\mathcal{D} = \{\tau_i\}$ with policy $\pi_\theta$.
  Compute $G_t$ and $G_t^\lambda = H_t^{\text{GAE}(\lambda)} + \hat{v}_\phi(S_t)$.
  Compute PPO advantage $H_t = H_t^{\text{GAE}(\lambda)}$ and normalize.
  **for** epoch $= 1, 2, \ldots$ **do**
    Slice trajectories $\mathcal{D}$ into mini-batches.
    **for** each mini-batch $B$ **do**
      Set $\hat{\rho}_t(\theta)$ by Eq. 8 and detach it from the computation graph.
      Reparameterize the action $A_t = f_\theta(\epsilon_t; S_t)$.
      Compute the predicted reward $\hat{R}_{t+1} = \hat{r}_w(S_t, f_\theta(\epsilon_t; S_t))$.
      Update $\theta$ by maximizing $\mathbb{E}_B[\hat{\rho}_t(\theta) H_t^{\text{RPG}}]$, where $H_t^{\text{RPG}} = \hat{R}_{t+1} + (\gamma G_{t+1}^\lambda - \hat{v}_\phi(S_t)) \times \log \pi_\theta(A_t|S_t)$.
      Update $\phi$ by minimizing $\mathbb{E}_B[(\hat{v}_\phi(S_t) - G_t)^2]$.
      Update $w$ by minimizing $\mathbb{E}_B[(\hat{r}_w(S_t, A_t) - R_{t+1})^2]$.
    **end for**
  **end for**
**end for**

approximate them. Furthermore, we use $\lambda$-return (Sutton and Barto, 2018) $G_{t+1}^\lambda$ to replace $v_{\pi_\theta}(s')$ in Eq. 6, which has a close relationship to GAE $H_t^{\text{GAE}(\lambda)}$ (Schulman et al., 2015b), i.e. $G_t^\lambda = H_t^{\text{GAE}(\lambda)} + v_{\pi_\theta}(s_t)$, where $\lambda \in [0, 1]$. Using $\lambda$-returns significantly reduce the variance of gradient estimations while retaining tolerable biases (Schulman et al., 2015b).

We first briefly describe PPO, which is the basis of the new RPG algorithm. PPO is simple to implement, closely related to on-policy learning, and achieves good performance on many tasks (Schulman et al., 2017). The key idea behind PPO is to constrain the policy update by using a clipped surrogate objective $\mathbb{E}_{\pi_{\theta_{\text{old}}}}[l_t(\theta)]$, where $l_t(\theta) = \min(\rho_t(\theta)H_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)H_t)$, where $H_t$ is an estimator of the advantage function, and $\rho_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the importance sampling ratio. Then the gradients of $l_t(\theta)$ can be written as

$$\nabla_\theta l_t(\theta) = \begin{cases} 0, & H_t > 0, \rho_t(\theta) > 1+\epsilon \\ 0, & H_t < 0, \rho_t(\theta) < 1+\epsilon \\ \nabla_\theta \rho_t(\theta) H_t, & \text{otherwise.} \end{cases} \quad (7)$$

Inspired by Eq. 7, we can use a modified ratio $\hat{\rho}_t(\theta)$,

$$\hat{\rho}_t(\theta) = \begin{cases} 0, & H_t > 0, \rho_t(\theta) > 1+\epsilon \\ 0, & H_t < 0, \rho_t(\theta) < 1+\epsilon \\ \rho_t(\theta), & \text{otherwise.} \end{cases} \quad (8)$$

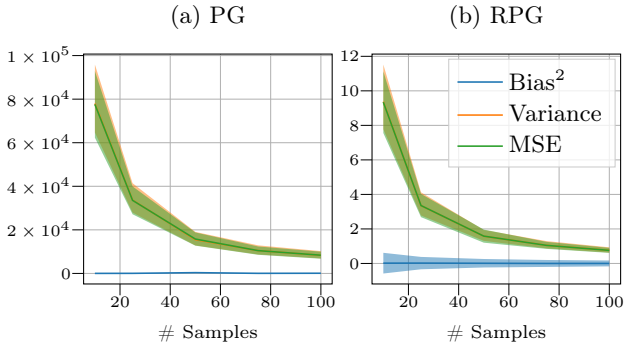



Figure 1: The bias, variance, and mean squared error (MSE) of the estimated gradient w.r.t. the number of samples for the PG estimator (left plot) and the RPG estimator (right plot). The shaded area representing a 95% interval using bootstrapping techniques. The values of bias, variance, and MSE for the RPG estimator are significantly smaller than the values for the PG estimator, which is clear from the Y-axis ranges.

Define $l_t(\theta) = \hat{\rho}_t(\theta)H_t$. It is easy to verify that this $\nabla_\theta l_t(\theta)$ is exactly the same as Eq. 7. For a complete algorithm description of PPO, please check the appendix.

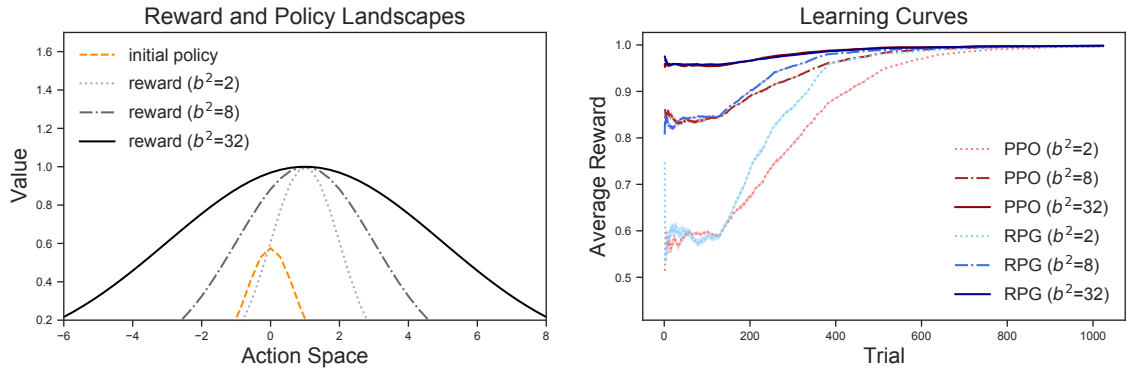
The RPG algorithm builds on PPO with two major modifications. First, we replace the original LR estimator in PPO with the RPG estimator. Second, in order to use reward gradients, we have a neural network to learn the reward function. Basically, RPG can be viewed as a version of PPO but using the RPG estimator to do gradient estimation. The detailed algorithm description for RPG is listed in Algorithm 1.

# 5 EXPERIMENTS

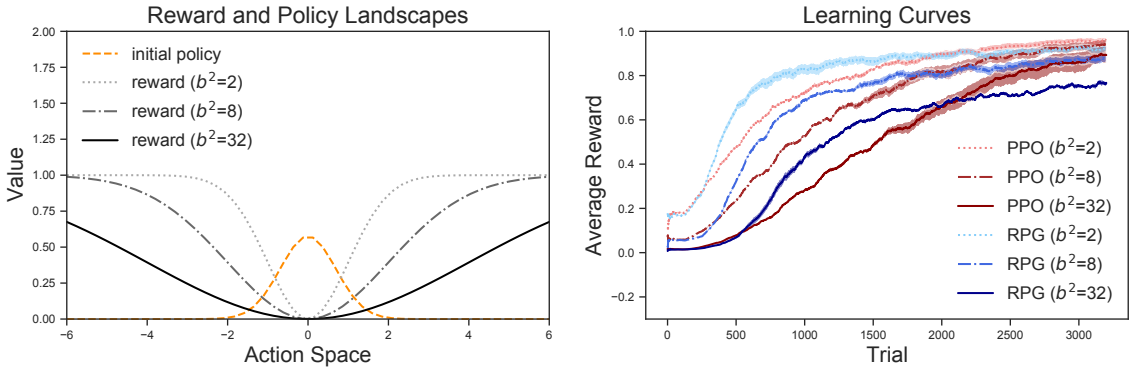
In this section, we first analyze the bias-variance tradeoff for the RPG estimator on a Linear Quadratic Gaussian (LQG) control task. We then gain more understanding to the advantages and drawbacks of using reward gradients on two bandit tasks. Furthermore, we investigate the benefit of RPG when the reward function is known. Finally, we evaluate our algorithm on six MuJoCo control tasks (Todorov et al., 2012) and one robot task compared to the baseline method.

## A Bias-variance Analysis of the RPG Estimator

Environments can have highly stochastic rewards (Brockman et al., 2016). Knowing the reward function in advance allows reducing the stochasticity of the gradient estimator. Furthermore,




(a) Peaks. RPG converges much faster than PPO in the whole training stage, when the variance is not so large.




(b) Holes. RPG converges much faster than PPO at the early training stage, under various variance settings. Then it slows down at the later stage.

Figure 2: The reward landscapes of Peaks and Holes as well as learning curves for PPO and RPG during training. The initial Gaussian policies are also visualized. All results were averaged over 30 runs; the shaded areas represent standard errors.

in most cases, reparameterization gradients exhibit lower variance w.r.t. likelihood ratio gradients (Xu et al., 2019). However, the variance of our estimator depends on many compounding factors (e.g. reward shapes and environment dynamics) that complicate a theoretical analysis. To compensate for this gap, we empirically investigate this aspect on a LQG problem.

The LQG control problem is one of the most classical control problems in control theory, with linear dynamics, quadratic reward, and Gaussian noise. Using the Riccati equations, we can solve the LQG problem in closed form. This property makes the LQG task an ideal benchmark to do bias-variance analysis for different policy gradient estimators. Specifically, we consider the discrete-time LQG problem, defined as

$$\max_{\theta} \sum_{t=0}^{\infty} \gamma^t r_t$$
$$\text{s.t. } \mathbf{s}_{t+1} = A\mathbf{s}_t + B\mathbf{a}_t; \quad r_t = -\mathbf{s}_t{}^\top Q \mathbf{s}_t - \mathbf{a}_t{}^\top Z \mathbf{a}_t$$
$$\mathbf{a}_{t+1} = \Theta \mathbf{s}_t + \Sigma \epsilon_t; \quad \epsilon_t \sim \mathcal{N}(0, I),$$

where $A, B, Q, Z, \Sigma$, and $\Theta$ are diagonal matrices and $\Theta = \text{diag}(\theta)$. Our policy is Gaussian: $\mathbf{a} \sim \mathcal{N}(\Theta\mathbf{s}; \Sigma)$, where $\theta \in \mathbb{R}^2$.

Given an LQG task, we study how the bias, the variance, and the mean squared error (MSE) of the estimated gradient vary w.r.t. the number of samples for both the policy gradient (PG) estimator and the RPG estimator. A similar study can also be found in Tosatto et al. (2021). Note that we do not subtract a baseline term in either estimator as in Eq. 6. In our experiment, we assume that the true reward function and the true value function are provided to both gradient estimators. Once the reward function is given, the RPG estimator is able to use not only the reward signals but also reward gradients. However, the PG estimator can only use the reward signals since it is not designed to use reward gradients. For further details, please refer to the appendix. In Figure 1, the left and right plots show the values of the PG estimator and the RPG estimator, respectively. Clearly, both the bias and the variance of the RPG estimator are significantly smaller than the ones of the PG estimator, which can be noticed from

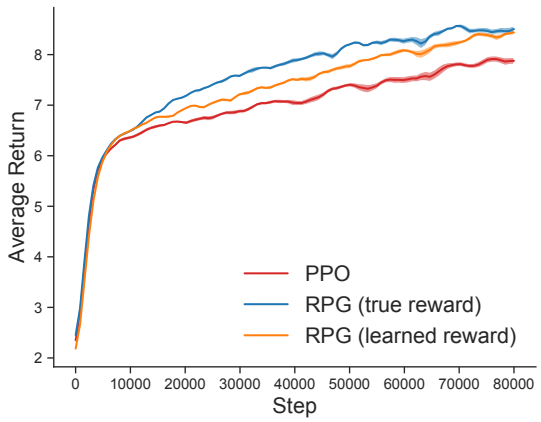



Figure 3: The learning curves for PPO and RPG during training on Mountain Climbing. The results were averaged over 20 runs, with the shaded area representing one standard error. In terms of convergence rate, RPG (true reward) > RPG (learned reward) > PPO.

the Y-axis ranges of the two plots. This result indicates that by incorporating reward gradients, the RPG estimator is able to reduce the bias and the variance of gradient estimation substantially.

## Benefits and Drawbacks of Reward Gradients

To further understand the role of reward gradients, we design two kinds of bandit tasks—*Peaks* and *Holes*—with continuous action spaces $\mathbb{R}$. The reward functions of Peaks and Holes are defined as $r(a) = \exp\left(-\frac{(a-1)^2}{b^2}\right)$ and $r(a) = 1 - \exp\left(-\frac{a^2}{b^2}\right)$, as shown in Figure 2(a) and Figure 2(b), respectively. In our experiments, a small ($0.01\times$) Gaussian noise is added to all reward signals and $b^2$ is chosen from $[2, 8, 32]$.

The initial policy distribution is a Gaussian distribution $\mathcal{N}(0, 0.69)$. For RPG, we use a neural network to approximate the true reward function, denoted by $\hat{r}_w(a)$. In bandit cases, there are no value functions and we have the following single-sample gradients for PPO and RPG, without considering normalization for simplicity:

$$\nabla_\theta l_t^{\text{PPO}}(\theta) = \hat{\rho}_t(\theta) R_t \nabla_\theta \log \pi_\theta(A_t) \text{ and}$$
$$\nabla_\theta l_t^{\text{RPG}}(\theta) = \hat{\rho}_t(\theta) \nabla_\theta f_\theta(\epsilon_t) \nabla_a \hat{r}_w(a)|_{a=A_t}, \quad (9)$$

where $\theta = [\mu, \sigma]$. We test PPO and RPG on these bandits over 30 runs. The gradient is clipped by 1.

The learning curves of PPO and RPG are visualized in Figure 2(a) and 2(b). We observe that as $b^2$ increases, the learning speed decreases for both PPO and RPG, on Peaks and Holes. This is because the magnitude of reward's gradient decreases as $b^2$ increases which reduces $\nabla_\theta l_t(\theta)$ and slows down the learning update, as suggested by Eq. 9.

Furthermore, RPG converges much faster than PPO on Peaks in the whole training period when $b^2$ is not so large. This shows that RPG is able to find the optimal action quicker than PPO, with the help of a relatively large reward's gradient. On Holes, RPG escapes from the sub-optimal action faster than PPO at the early training stage, but it slows down later and performs worse than PPO in the end. This is not surprising since the reward function is still sharp at the early training stage (when $|a|$ is small) but tends to become flat later (i.e. $|\nabla_a \hat{r}_w(a)|$ is smaller.) which slows down the learning process of RPG.

Therefore, we conclude that in bandits the reward's gradient accelerates learning when it is relatively large but hurts learning when the reward function is too flat. Reward and action-value gradients are identical in bandits; to compensate that, and we conduct the rest of the studies using simple and complex MDPs.

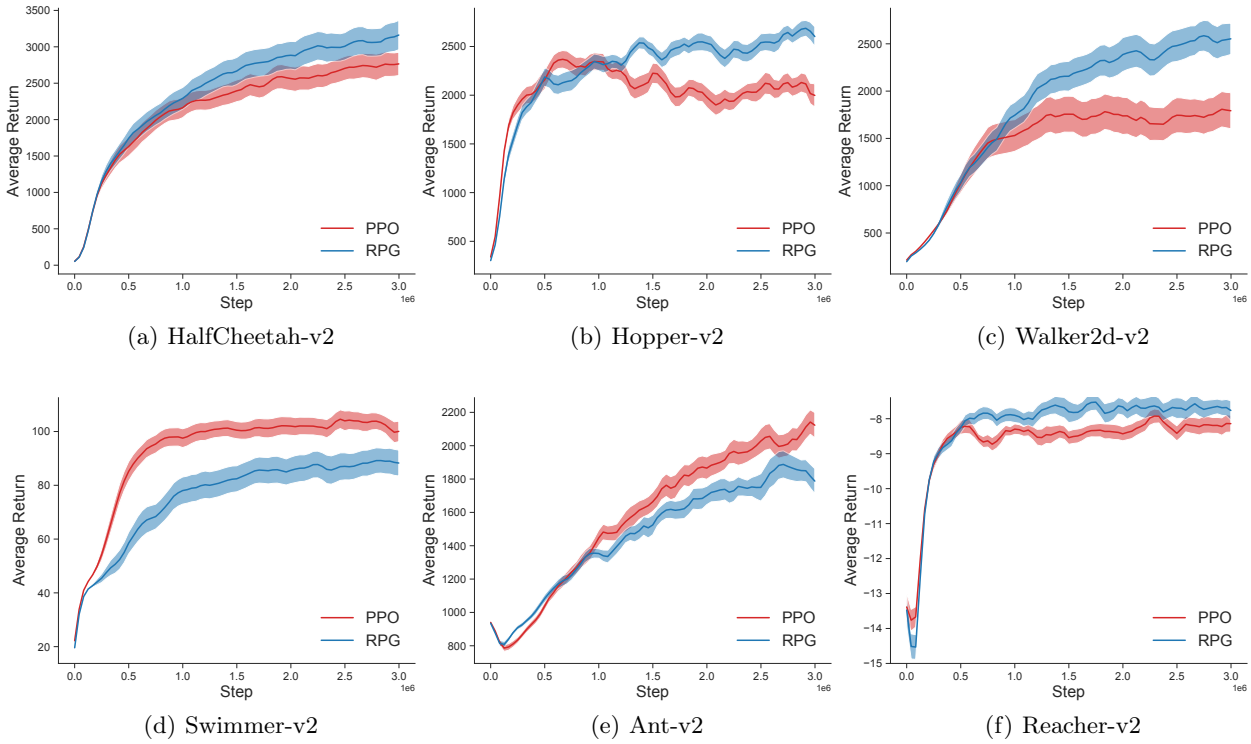
## The Benefit of Knowing the Reward Function

To explore the role of reward gradients in MDP settings, we design a simple MDP—*Mountain Climbing*—with continuous state and action spaces. Specifically, $\mathcal{S} = [-8, 8]^2$, $\mathcal{A} = [-1, 1]^2$, $S_0 = (0, 0), R_{t+1} = r(S_t, A_t)$, and $S_{t+1} = S_t + A_t + \epsilon$ where $r(s, a) = \exp(-||s + a - \nu||_2^2)$, $\epsilon \sim \mathcal{U}(-0.005, 0.005)$, and $\nu = (1, -1)$. Every episode ends in 10 steps. We test PPO (LR gradient) and RPG (LR gradient + reward gradient) on this MDP for $80,000$ steps. We set the number of epochs in PPO and RPG to 1 and the batch and mini-batch size to 40. The gradient is clipped by 0.5.

The average returns over 20 runs for each algorithm during training are shown in Figure 3. By utilizing reward gradients, RPG (LR gradient + reward gradient) outperforms PPO (LR gradient) significantly. Moreover, the reward gradients of the true reward function are more helpful to accelerate learning than the reward gradients of a learned one, probably due to a higher accuracy of gradient estimation; although two versions of RPG reach a similar performance in the end.

## Evaluation on Simulated Benchmark Tasks

To further evaluate our algorithm, we measure its performance on six MuJoCo control tasks (Todorov, 2014)




(a) HalfCheetah-v2 (b) Hopper-v2 (c) Walker2d-v2

(d) Swimmer-v2 (e) Ant-v2 (f) Reacher-v2

Figure 4: Learning curves of evaluations on six benchmark tasks for PPO and RPG. The results were averaged over 30 runs, with the shaded area representing one standard error.

through OpenAI Gym (Brockman et al., 2016). Our PPO implementation is based on Zhang (2018) and Achiam (2018). Then we implement RPG based on this version of PPO. The appendix contains all hyper-parameters which are largely borrowed from some popular implementations. The code is also submitted as a supplementary file. For the simulated tasks, each algorithm was trained on every task for 3 million steps. For every 5 epochs, we evaluated the agent's test performance using a deterministic policy for one episode. Our results were reported by averaging over 30 runs with different random seeds. The learning curves during evaluation are presented in Figure 4. Overall, RPG outperforms PPO significantly on three tasks – HalfCheetah, Hopper, and Walker2d. It is slightly better on Reacher and worse on Ant, compared to PPO. On Swimmer, however, PPO has a clear advantage.

### Evaluation on a Real-Robot Task

We include results on a real-robot task to showcase RPG's ability to effectively learn in a real-world scenario. We use the Real-Robot Reacher task based on a UR robotic arm in which an agent must move the fingertip of a robot arm as close and as fast as possible to a randomly selected target by rotating the base and elbow joints. Each episode is 4 seconds long followed by a reset, bringing the fingertip to a start position. The appendix contains the full description of the task. RPG runs on Real-Robot Reacher for 90,000 time steps at 40ms action-cycle time, resulting in an hour of robot-experience time excluding resets. The undiscounted episodic returns were stored during training without measuring a separate test performance.

Figure 5 shows the resulting learning curves averaged over five independent runs. RPG continues to increase performance throughout learning. For comparison, we also included the performance of PPO, which is known to learn an effective reaching behavior on this task, specifically at this performance level of about 300 average return (Mahmood et al., 2018; Farrahi and Mahmood, 2020). Both RPG and PPO achieved the same performance on this task, which is not surprising as we did not perform a hyper-parameter search. Instead, our results confirm that RPG is at least as effective as PPO on a well-studied real-robot task.

## 6 DISCUSSION

Our work focuses more on understanding the properties of the RPG estimator by conducting a series of analysis on simple environments. Moreover, the RPG theorem only provides a new way to estimate policy

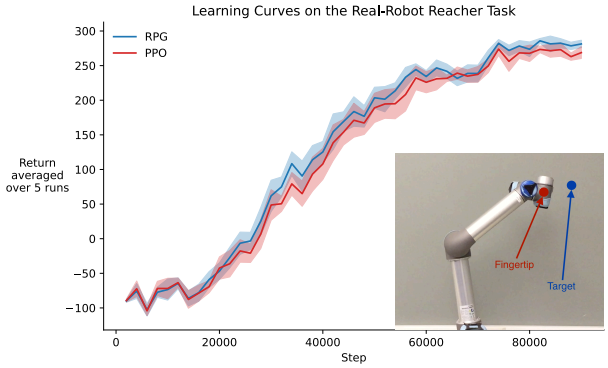



Figure 5: The learning curves for RPG and PPO on the Real-Robot Reacher task. The results were averaged over 5 runs, with the shaded area representing one standard error. The bottom-right image depicts the task setup. RPG is as effective as PPO on this task.

gradient; the implementation of actual algorithms can vary. For example, the RPG version of the naïve actor-critic would be a straight-forward implementation. Our current implementation builds on PPO, and it benefits from PPO's techniques as well. To further explore and exploit the advantage of the RPG estimator, we may develop more advanced implementations by combining some modern techniques in the future, such as entropy regularization (Haarnoja et al., 2018), parallel training (Mnih et al., 2016), separating training phases for policy and value functions (Cobbe et al., 2021), Retrace (Munos et al., 2016), and V-trace (Espeholt et al., 2018), etc. Our current implementation is just one approach; exploring more possibilities are among the potential future works resulting from this work.

# 7 CONCLUSION

In this paper, we introduced a novel strategy to compute the policy gradient which uses reward gradients without a model. Based on this strategy, we developed—RPG—a new on-policy policy gradient algorithm. We showed that our method of using reward gradients is beneficial over the PG estimator in terms of the bias-variance trade-off and sample efficiency. Experiments showed that RPG generally outperformed PPO on several simulation tasks. RPG is limited to the same class of tasks where reparameterization applies and, thus, not directly applicable to tasks with discrete actions. However, combined with the Gumbel-Softmax technique (Jang et al., 2017), it is possible to apply RPG on discrete control tasks as well.

## Acknowledgements

We gratefully acknowledge funding from the Canada CIFAR AI Chairs program, the Reinforcement Learning and Artificial Intelligence (RLAI) laboratory, the Alberta Machine Intelligence Institute (Amii), and the Natural Sciences and Engineering Research Council (NSERC) of Canada. We also thankfully acknowledge the donation of the UR5 arm from the Ocado Group.

## References

Achiam, J. (2018). Spinning Up in deep reinforcement learning.

Afsar, M. M., Crump, T., and Far, B. (2021). Reinforcement learning based recommender systems: A survey. *arXiv preprint arXiv:2101.06286*.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

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

Cai, Q., Pan, L., and Tang, P. (2020). Deterministic value-policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3316–3323.

Carvalho, J., Tateo, D., Muratore, F., and Peters, J. (2021). An empirical analysis of measure-valued derivatives for policy gradients. In *2021 International Joint Conference on Neural Networks*. IEEE.

Cobbe, K. W., Hilton, J., Klimov, O., and Schulman, J. (2021). Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K. O., and Clune, J. (2018). Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *NeurIPS*.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley,

T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR.

Farrahi, H. and Mahmood, A. R. (2020). Making hyperparameters of proximal policy optimization robust to time discretization. *Robot Learning Workshop at NeurIPS*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR.

Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, University of Cambridge.

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. (2015). Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, volume 28, pages 2944–2952.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *Journal of Machine Learning Research*, 14(5).

Imani, E., Graves, E., and White, M. (2018). An off-policy policy gradient theorem using emphatic weightings. *Advances in Neural Information Processing Systems*, 31:96–106.

Jafferjee, T., Imani, E., Talvitie, E., White, M., and Bowling, M. (2020). Hallucinating value: A pitfall of Dyna-style planning with imperfect environment models. *arXiv preprint arXiv:2006.04363*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

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

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2020). Monte Carlo gradient estimation in machine learning. *Journal of Machine Learning Research*, 21(132):1–62.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 1054–1062.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. (2018). PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*.

Pflug, G. C. (1989). Sampling derivatives of probabilities. *Computing*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1):123–158.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press, second edition.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Todorov, E. (2014). Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Tosatto, S., Carvalho, J., and Peters, J. (2021). Batch reinforcement learning with a nonparametric off-policy policy gradient. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

van Hasselt, H. P., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32:14322–14333.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, H., Zheng, S., Xiong, C., and Socher, R. (2019). On the generalization gap in reparameterizable reinforcement learning. In *International Conference on Machine Learning*, pages 6648–6658.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Xu, M., Quiroz, M., Kohn, R., and Sisson, S. A. (2019). Variance reduction properties of the reparameterization trick. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2711–2720.

Zha, D., Xie, J., Ma, W., Zhang, S., Lian, X., Hu, X., and Liu, J. (2021). DouZero: Mastering DouDizhu with self-play deep reinforcement learning. *arXiv preprint arXiv:2106.06135*.

Zhang, S. (2018). Modularized implementation of deep RL algorithms in PyTorch. `https://github.com/ShangtongZhang/DeepRL`.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., and Tang, J. (2018). Deep reinforcement learning for page-wise recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 95–103.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., and Li, Z. (2018). DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176.

# Supplementary Material: Model-free Policy Learning with Reward Gradients

# A THEOREMS AND PROOFS

## A.1 Proof of Reward Policy Gradient Theorem

Let $d^{\pi_\theta}(s') = \int \sum_{t=0}^{\infty} \gamma^t p_0(s) p(s \to s', t, \pi_\theta) \mathrm{d}s$ be the (discounted) stationary state distribution for policy $\pi_\theta$; and $p(s \to s', t, \pi_\theta)$ the transition probability from $s$ to $s'$ with $t$ steps under policy $\pi_\theta$.

**Theorem 1** (Reward Policy Gradient Theorem)**.** *Suppose that the MDP satisfies Assumption 1 and 2, then*

$$\nabla_\theta J(\theta) = \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\left[\nabla_\theta r(s, f_\theta(\epsilon; s))|_{\epsilon=g_\theta(a;s)} + \gamma v_{\pi_\theta}(s')\nabla_\theta \log \pi_\theta(a|s)\right] \mathrm{d}s\mathrm{d}a\mathrm{d}s'.$$

*Proof.* By the policy gradient theorem, we have

$$\nabla_\theta J(\theta) = \int d^{\pi_\theta}(s)\pi_\theta(a|s) q_{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}a\mathrm{d}s.$$

Next, we split the action-value function $q_{\pi_\theta}(s,a)$ into two parts:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int d^{\pi_\theta}(s)\pi_\theta(a|s)\left[r(s,a) + \gamma \int p(s'|s,a) v_{\pi_\theta}(s') \mathrm{d}s'\right] \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}a\mathrm{d}s \\
&= \int d^{\pi_\theta}(s)\pi_\theta(a|s) r(s,a) \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s) r(s,a) \nabla_\theta \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s) \nabla_\theta \left(\int \pi_\theta(a|s) r(s,a) \mathrm{d}a\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s'.
\end{aligned}$$

Now, we apply the reparameterization technique to the first part,

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int d^{\pi_\theta}(s) \nabla_\theta \left(\int \pi_\theta(a|s) r(s,a) \mathrm{d}a\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s) \nabla_\theta \left(\int p(\epsilon) r(s, f_\theta(\epsilon; s)) \mathrm{d}\epsilon\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s) \left(\int p(\epsilon) \nabla_\theta r(s, f_\theta(\epsilon; s)) \mathrm{d}\epsilon\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s'.
\end{aligned}$$

We then apply the reverse operation of reparameterization to the first part,

$$\begin{aligned}
&\nabla_\theta J(\theta) \\
&= \int d^{\pi_\theta}(s) \left(\int p(\epsilon) \nabla_\theta r(s, f_\theta(\epsilon; s)) \mathrm{d}\epsilon\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s) \left(\int \pi_\theta(a|s) \nabla_\theta r(s, f_\theta(\epsilon; s))|_{\epsilon=g_\theta(a;s)} \mathrm{d}a\right) \mathrm{d}s + \gamma \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a) v_{\pi_\theta}(s') \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}s\mathrm{d}a\mathrm{d}s' \\
&= \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\left[\nabla_\theta r(s, f_\theta(\epsilon; s))|_{\epsilon=g_\theta(a;s)} + \gamma v_{\pi_\theta}(s')\nabla_\theta \log \pi_\theta(a|s)\right] \mathrm{d}s\mathrm{d}a\mathrm{d}s'.
\end{aligned}$$

□

## A.2 Proof of Reward Policy Gradient with Function Approximation

**Theorem 2** (Reward Policy Gradient with Function Approximation)**.** *Consider a parametric approximation of the reward function $\hat{r}_\omega(s,a)$ and a parametric approximation of the value function $\hat{v}_\phi(s)$ such that*

1. $\nabla_a \hat{r}_\omega(s,a) = \nabla_\theta^\intercal f_\theta(\epsilon;s)|_{\epsilon=g_\theta(a;s)}\omega$,
2. $\int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\nabla_\phi \hat{v}_\phi(s')\mathrm{d}s\mathrm{d}a = \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)\nabla_\theta \log\pi_\theta(a|s)\mathrm{d}s\mathrm{d}a$,

*where* $\phi = \arg\min_\phi \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} \left[(\hat{v}_\phi(s') - v(s'))^2\right]$
*and* $\omega = \arg\min_\omega \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} [(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))^\intercal(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))]$.
*Then,* $\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} \left[\nabla_\theta \hat{r}_\omega(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)} + \gamma\hat{v}_\phi(s')\nabla_\theta\log\pi_\theta(a|s)\right]$.

*Proof.* By the definitions of $\phi$ and $\omega$, we have

$$
\begin{aligned}
&\omega = \arg\min_\omega \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} [(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))^\intercal(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))]\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} [(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))\nabla_\omega\nabla_a \hat{r}_\omega(s,a)] = 0\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} \left[(\nabla_a \hat{r}_\omega(s,a) - \nabla_a r(s,a))\nabla_\theta f_\theta(\epsilon;s)|_{\epsilon=g(a;s)}\right] = 0\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} \left[\nabla_a \hat{r}_\omega(s,a)\nabla_\theta f_\theta(\epsilon;s)|_{\epsilon=g(a;s)}\right] = \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} \left[\nabla_a r(s,a)\nabla_\theta f_\theta(\epsilon;s)|_{\epsilon=g(a;s)}\right]\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} \left[\nabla_\theta \hat{r}_\omega(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)}\right] = \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta}} \left[\nabla_\theta r(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)}\right],
\end{aligned}
$$

and

$$
\begin{aligned}
&\phi = \arg\min_\phi \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} \left[(\hat{v}_\phi(s') - v(s'))^2\right]\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} [(\hat{v}_\phi(s') - v(s'))\nabla_\phi \hat{v}_\phi(s')] = 0\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} [(\hat{v}_\phi(s') - v(s'))\nabla_\theta \log\pi_\theta(a|s)] = 0\\
\implies &\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} [\hat{v}_\phi(s')\nabla_\theta \log\pi_\theta(a|s)] = \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} [v(s')\nabla_\theta \log\pi_\theta(a|s)].
\end{aligned}
$$

Combine the above results with Theorem 1, we have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} \left[\nabla_\theta r(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)} + \gamma v(s')\nabla_\theta\log\pi_\theta(a|s)\right]\\
&= \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi_\theta}\\ a\sim\pi_\theta\\ s'\sim p(\cdot|s,a)}} \left[\nabla_\theta \hat{r}_\omega(s, f_\theta(\epsilon;s))|_{\epsilon=g_\theta(a;s)} + \gamma\hat{v}_\phi(s')\nabla_\theta\log\pi_\theta(a|s)\right].
\end{aligned}
$$

□

## A.3 Other Related Theorems

**Theorem 3** (Reparameterization Policy Gradient Theorem)**.** *Suppose that the MDP satisfies Assumption 1 and 2, then*

$$\nabla_\theta J(\theta) = \int d^{\pi_\theta}(s) p(\epsilon) \nabla_\theta f_\theta(\epsilon; s) \nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \mathrm{d}\epsilon \mathrm{d}s.$$

*Proof.* By the policy gradient theorem, we have

$$\nabla_\theta J(\theta) = \int d^{\pi_\theta}(s) \pi_\theta(a|s) q_{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}a \mathrm{d}s.$$

Thus

$$\begin{aligned}
&\nabla_\theta J(\theta) \\
&= \int d^{\pi_\theta}(s) \pi_\theta(a|s) q_{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s) \mathrm{d}a \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \left( \int q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s) \mathrm{d}a \right) \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \left[ \int \nabla_\theta (q_{\pi_\theta}(s,a) \pi_\theta(a|s)) \mathrm{d}a - \int \pi_\theta(a|s) \nabla_\theta q_{\pi_\theta}(s,a) \mathrm{d}a \right] \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \left[ \nabla_\theta \left( \int q_{\pi_\theta}(s,a) \pi_\theta(a|s) \mathrm{d}a \right) - \int \pi_\theta(a|s) \nabla_\theta q_{\pi_\theta}(s,a) \mathrm{d}a \right] \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \left[ \nabla_\theta \left( \int p(\epsilon) q_{\pi_\theta}(s, f_\theta(\epsilon; s)) \mathrm{d}\epsilon \right) - \int p(\epsilon) \nabla_\theta q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \mathrm{d}\epsilon \right] \mathrm{d}s \quad \text{(by reparameterization)} \\
&= \int d^{\pi_\theta}(s) \left[ \int p(\epsilon) \left( \nabla_\theta f_\theta(\epsilon; s) \nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} + \nabla_\theta q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \right) \mathrm{d}\epsilon - \int p(\epsilon) \nabla_\theta q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \mathrm{d}\epsilon \right] \mathrm{d}s \\
&= \int d^{\pi_\theta}(s)\ p(\epsilon) \nabla_\theta f_\theta(\epsilon; s) \nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \mathrm{d}\epsilon \mathrm{d}s.
\end{aligned}$$

□

**Remark 2.** *This theorem is a direct application of reparameterization to the gradient of the policy objective. It is understood to be known but is not formally presented or derived for policy gradients in any existing work. We include it here for completeness as well as a useful theoretical tool.*

**Corollary 1** (Deterministic Policy Gradient Theorem)**.** *Suppose that the MDP satisfies Assumption 1 and 2, for a deterministic policy $\mu_\theta(s)$, we have*

$$\nabla_\theta J(\theta) = \int d^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a q_{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s.$$

*Proof.* Let $p(\epsilon)$ be the delta function. Thus $\int_\epsilon p(\epsilon) f_\theta(\epsilon; s) \mathrm{d}\epsilon = f_\theta(0; s)$. Furthermore, let $f_\theta(0; s) = \mu_\theta(s)$. By Theorem 3,

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int d^{\pi_\theta}(s) \left( \int p(\epsilon) \nabla_\theta f_\theta(\epsilon; s) \nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} \mathrm{d}\epsilon \right) \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \nabla_\theta f_\theta(0; s) \nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(0;s)} \mathrm{d}s \\
&= \int d^{\pi_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a q_{\pi_\theta}(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s.
\end{aligned}$$

□

**Remark 3.** *Both DPG and DDPG are proposed based on this theorem, which is first proved by Silver et al. (2014). It is limited to deterministic policies and thus can be deduced as a corollary of Theorem 3, which is applicable to stochastic policies as well.*

**Corollary 2** (Entropy-regularized Reparameterization Policy Gradient Theorem)**.** *Consider the entropy-regularized values* $v_{\pi_\theta}(s_0) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t(r(s_t,a_t)+\alpha\mathcal{H}(\pi_\theta(\cdot|s_t)))\right]$ *and* $q_{\pi_\theta}(s_0,a_0) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)+\alpha\sum_{t=1}^{\infty}\gamma^t\mathcal{H}(\pi_\theta(\cdot|s_t))\right]$*, where* $\mathcal{H}(p) = -\int_x p(x)\log p(x)\mathrm{d}x$ *is the differential entropy for probability density function* $p(x)$*, and* $\alpha$ *is a positive constant. Suppose that the MDP satisfies Assumption 1 and 2, then*

$$\nabla_\theta J(\theta) = \int d^{\pi_\theta}(s)p(\epsilon)\left[\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} - \alpha\nabla_\theta\log\pi_\theta(f_\theta(\epsilon;s)|s)\right]\mathrm{d}\epsilon\mathrm{d}s.$$

*Proof.* By definition, we have

$$v_{\pi_\theta}(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t(r(s_t,a_t)+\alpha\mathcal{H}(\pi_\theta(\cdot|s_t)))\mid s_0=s\right],$$
$$q_{\pi_\theta}(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)+\alpha\sum_{t=1}^{\infty}\gamma^t\mathcal{H}(\pi_\theta(\cdot|s_t))\mid s_0=s,a_0=a\right],$$

where $\mathcal{H}(p) = -\int_x p(x)\log p(x)\mathrm{d}x$ is the differential entropy for probability density function $p(x)$.

Then $v_{\pi_\theta}$ and $q_{\pi_\theta}$ are connected by

$$\begin{aligned}v_{\pi_\theta}(s) &= \mathbb{E}_\pi[q_{\pi_\theta}(s,a)] + \alpha\mathcal{H}(\pi_\theta(\cdot|s))\\ &= \int\pi_\theta(a|s)(q_{\pi_\theta}(s,a)-\alpha\log\pi_\theta(a|s))\mathrm{d}a\\ &= \int p(\epsilon)\left(q_{\pi_\theta}(s,f_\theta(\epsilon;s))-\alpha\log\pi_\theta(f_\theta(\epsilon;s)|s)\right)\mathrm{d}\epsilon,\end{aligned}$$

and the Bellman equation for $q_{\pi_\theta}$ is

$$\begin{aligned}q_{\pi_\theta}(s,a) &= \mathbb{E}_\pi\left[r(s,a)+\gamma(q_{\pi_\theta}(s',a')+\alpha\mathcal{H}(\pi_\theta(\cdot|s')))\right]\\ &= r(s,a)+\gamma\mathbb{E}[v_{\pi_\theta}(s')] = r(s,a)+\gamma\int p(s'|s,a)v_{\pi_\theta}(s')\mathrm{d}s'.\end{aligned}$$

Then

$$\begin{aligned}&\nabla_\theta v_{\pi_\theta}(s)\\ =&\nabla_\theta\left(\int\pi_\theta(a|s)q_{\pi_\theta}(s,a)\mathrm{d}a\right)+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\nabla_\theta\left(\int p(\epsilon)q_{\pi_\theta}(s,f_\theta(\epsilon;s))\mathrm{d}\epsilon\right)+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\int p(\epsilon)\nabla_\theta q_{\pi_\theta}(s,f_\theta(\epsilon;s))\mathrm{d}\epsilon+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\int p(\epsilon)\left(\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}+\nabla_\theta q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\right)\mathrm{d}\epsilon+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\int p(\epsilon)\left(\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}+\gamma\int p(s'|s,f_\theta(\epsilon;s))\nabla_\theta v_{\pi_\theta}(s')\mathrm{d}s'\right)\mathrm{d}\epsilon+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon+\gamma\int p(\epsilon)p(s'|s,f_\theta(\epsilon;s))\nabla_\theta v_{\pi_\theta}(s')\mathrm{d}\epsilon\mathrm{d}s'+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s))\\ =&\int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon+\gamma\int p(s\to s',1,\pi_\theta)\nabla_\theta v_{\pi_\theta}(s')\mathrm{d}s'+\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)),\end{aligned}$$

where $p(s\to s',1,\pi_\theta) = \int p(\epsilon)p(s'|s,f_\theta(\epsilon;s))\mathrm{d}\epsilon$.

Now iterating this formula we have

$$
\begin{aligned}
&\nabla_\theta v_{\pi_\theta}(s) \\
=&\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)) + \int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon + \gamma\int p(s\to s',1,\pi_\theta)\nabla_\theta v_{\pi_\theta}(s')\mathrm{d}s' \\
=&\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)) + \int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon + \gamma\int p(s\to s',1,\pi_\theta) \\
&\left(\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s')) + \int p(\epsilon')\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')}\mathrm{d}\epsilon' + \gamma\int p(s'\to s'',1,\pi_\theta)\nabla_\theta v_{\pi_\theta}(s'')\mathrm{d}s''\right)\mathrm{d}s' \\
=&\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)) + \int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon \\
&+\alpha\gamma\int p(s\to s',1,\pi_\theta)\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s'))\mathrm{d}s' + \gamma\int p(s\to s',1,\pi_\theta)p(\epsilon')\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')}\mathrm{d}\epsilon'\mathrm{d}s' \\
&+\gamma^2\int p(s\to s',1,\pi_\theta)p(s'\to s'',1,\pi_\theta)\nabla_\theta v_{\pi_\theta}(s'')\mathrm{d}s''\mathrm{d}s' \\
=&\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)) + \int p(\epsilon)\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)}\mathrm{d}\epsilon \\
&+\alpha\gamma\int p(s\to s',1,\pi_\theta)\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s'))\mathrm{d}s' + \gamma\int p(s\to s',1,\pi_\theta)p(\epsilon')\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')}\mathrm{d}\epsilon'\mathrm{d}s' \\
&+\gamma^2\int p(s\to s',2,\pi_\theta)\nabla_\theta v_{\pi_\theta}(s')\mathrm{d}s' \\
=&\dots \\
=&\int\sum_{t=0}^{\infty}\gamma^t p(s\to s',t,\pi_\theta)\left(\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s')) + \int p(\epsilon')\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')}\mathrm{d}\epsilon'\right)\mathrm{d}s',
\end{aligned}
$$

where $p(s\to s'',t+1,\pi_\theta) = \int p(s\to s',t,\pi_\theta)p(s'\to s'',1,\pi_\theta)\mathrm{d}s'$.

Furthermore,

$$
\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s)) = -\nabla_\theta\int\pi_\theta(a|s)\log\pi_\theta(a|s)\mathrm{d}a = -\nabla_\theta\int p(\epsilon)\log\pi_\theta(f_\theta(\epsilon;s)|s)\mathrm{d}\epsilon = -\int p(\epsilon)\nabla_\theta\log\pi_\theta(f_\theta(\epsilon;s)|s)\mathrm{d}\epsilon.
$$

Then

$$
\begin{aligned}
\nabla_\theta v_{\pi_\theta}(s) &= \int\sum_{t=0}^{\infty}\gamma^t p(s\to s',t,\pi_\theta)\left(\alpha\nabla_\theta\mathcal{H}(\pi_\theta(\cdot|s')) + \int p(\epsilon')\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')}\mathrm{d}\epsilon'\right)\mathrm{d}s' \\
&= \int\sum_{t=0}^{\infty}\gamma^t p(s\to s',t,\pi_\theta)p(\epsilon')\left(\nabla_\theta f_\theta(\epsilon';s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon';s')} - \alpha\nabla_\theta\log\pi_\theta(f_\theta(s,\epsilon')|s)\right)\mathrm{d}\epsilon'\mathrm{d}s'.
\end{aligned}
$$

Finally,

$$
\begin{aligned}
\nabla_\theta J(\theta) =&\nabla_\theta\int p_0(s)v_{\pi_\theta}(s)\mathrm{d}s = \int p_0(s)\nabla_\theta v_{\pi_\theta}(s)\mathrm{d}s \\
=&\int\sum_{t=0}^{\infty}\gamma^t p_0(s)p(s\to s',t,\pi_\theta)p(\epsilon)\left(\nabla_\theta f_\theta(\epsilon;s')\nabla_a q_{\pi_\theta}(s',a)|_{a=f_\theta(\epsilon;s')} - \alpha\nabla_\theta\log\pi_\theta(f_\theta(\epsilon;s)|s)\right)\mathrm{d}\epsilon\mathrm{d}s'\mathrm{d}s \\
=&\int d^{\pi_\theta}(s)\,p(\epsilon)\left(\nabla_\theta f_\theta(\epsilon;s)\nabla_a q_{\pi_\theta}(s,a)|_{a=f_\theta(\epsilon;s)} - \alpha\nabla_\theta\log\pi_\theta(f_\theta(\epsilon;s)|s)\right)\mathrm{d}\epsilon\mathrm{d}s.
\end{aligned}
$$

□

**Remark 4.** *There is a similar result presented by Haarnoja et al. (2018) for SAC. They obtain it by minimizing the Kullback-Leibler divergence between the new policy and the policy derived from the exponential of the soft Q-function. However, we derive it by directly minimizing the objective with the help of the RP gradient. This corollary provides an alternative way to understand SAC.*

# B HYPER-PARAMETER SETTINGS FOR SIMULATION TASKS

All our experiments were performed only with CPUs (AMD EPYC 7601 32-Core Processor).

## B.1 LQG

The parameters used for the LQG task are

$$A = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}; \quad B = \begin{bmatrix} 1e-4 & 0 \\ 0 & 1e-4 \end{bmatrix}; \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}; \quad \mathbf{s}_0 = [0.5, 0.5].$$

The discount factor is $\gamma = 0.99$. The number of steps for each episode is 100. The policy parameter is chosen randomly to be $\theta = [-1.1104430687690852, -1.3649958298432607]$.

## B.2 MuJoCo

Table 1: The hyper-parameter settings for PPO and RPG on MuJoCo tasks.

| **Hyper-parameter** | **PPO** | **RPG** |
|---|---|---|
| Policy network LR | $3 \times 10^{-4}$ | $3 \times 10^{-4}$ |
| Value network LR | $10^{-3}$ | $10^{-3}$ |
| Reward network LR | None | $10^{-3}$ |
| Hidden layers | [64, 64] | [64, 64] |
| Optimizer | Adam | Adam |
| Time-steps per iteration | 2048 | 2028 |
| Number of epochs | 10 | 10 |
| Mini-batch size | 64 | 64 |
| Discount factor ($\gamma$) | 0.99 | 0.99 |
| GAE parameter ($\lambda$) | 0.95 | 0.95 |
| PPO Clipping ($\epsilon$) | 0.2 | 0.2 |
| Target KL divergence | 0.01 | 0.01 |
| State Clipping | [-10, 10] | [-10, 10] |
| Gradient clipping | 2 | 2 |

**Algorithm 2** PPO

Input: initial policy parameters $\theta$, initial value estimate parameters $\phi$.
**for** $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D} = \{\tau_i\}$ by running policy $\pi_\theta$ in the environment.
  Compute returns $G_t$.
  Compute advantage estimates $H_t = H_t^{\mathrm{GAE}(\lambda)}$.
  Normalize $H_t$ using the sample mean and standard deviation of all advantage estimates.
  **for** epoch $= 0, 1, 2, \ldots$ **do**
    Shuffle and slice trajectories $\mathcal{D}$ into mini-batches.
    **for** each mini-batch $B$ **do**
      Set $\hat{\rho}_t(\theta)$ according to Equation 8.
      Update policy parameters $\theta$ by maximizing the objective: $\mathbb{E}_B[\hat{\rho}_t(\theta) H_t]$.
      Update value estimate parameters $\phi$ by minimizing: $\mathbb{E}_B[(\hat{v}_\phi(S_t) - G_t)^2]$.
    **end for**
  **end for**
**end for**

# C REAL-ROBOT TASK DESCRIPTION

For the real-robot experiment in this paper, we adopted the *UR-Reacher-2* task from Mahmood et al. (2018). The task runs on a UR5 robot arm from Universal Robots. By moving only two joints of the robot—the base and the elbow joints—an agent can move the robot around in two dimensions above a table to which the base joint is attached. The elbow joint connects the base joint to the fingertip.

Each episode lasts for 100 time steps of 40ms duration or 4 seconds in total. At the start of each episode the fingertip is reset to a fixed position above the robot, and a random target is drawn uniformly from a rectangular area around the fingertip. The goal of the agent is to get the fingertip as close and as fast as possible to the target by moving the base and elbow joints. The reward at each time step has two terms. The first term is the negative of the distance from fingertip to target. The second term rewards the agent equal to a Gaussian function that has its maximum value when the fingertip is on the target and reduces gradually as the fingertip moves away. In other words, the Gaussian function is centered at the target and evaluated at the distance from fingertip to target. This second reward term is used to encourage precision when fingertip and target are close by. The observation vector includes the angular position and velocity of the base and elbow joints, the vector from the fingertip to the target, and the previous action. The action vector contains the target angular velocity that the base and elbow joints should be set to.

In the original environment, whenever the learning agent pushed the fingertip outside a 2-dimensional rectangular boundary above the robot, a manually written script would move the fingertip to be inside the boundary. To reduce the frequency of scripted position corrections, we relaxed their movement constraints to a much larger area similarly to Farrahi and Mahmood (2020). The fingertip can now move freely as long as it is not too close to the table. The base and elbow joints can now rotate farther as well. We additionally made a few changes to the environment's code, without changing the task setup, to eliminate the protective stops that occurred when operating the joints at high speeds.