Policy Gradient with Kernel Quadrature

Anonymous authors
Paper under double-blind review

Abstract

Reward evaluation of episodes becomes a bottleneck in a broad range of reinforcement learning tasks. Our aim in this paper is to select a small but representative subset of a large batch of episodes, only on which we actually compute rewards for more efficient policy gradient iterations. We build a Gaussian process modeling of discounted returns or rewards to derive a positive definite kernel on the space of episodes, run an “episodic" kernel quadrature method to compress the information of sample episodes, and pass the reduced episodes to the policy network for gradient updates. We present the theoretical background of this procedure as well as its numerical illustrations in MuJoCo tasks.

1 Introduction

Reinforcement learning (RL) aims to learn a policy model that maximizes the cumulative average of rewards (Sutton & Barto, 2018). Policy gradient algorithms operate based on gradient ascent in the policy parameter space (Gullapalli, 1990; Williams, 1992; Schulman et al., 2017) and have greatly benefited from the latest advancements in neural network models. These algorithms have found widespread applications in domains such as robotics (Peters & Schaal, 2008), large language models (Ouyang et al., 2022), medical diagnosis (Xia et al., 2020), and many others.

Despite the broad applications of RL, a significant challenge, often overlooked, is the extensive computational or monetary cost associated with reward evaluations in real-world RL scenarios. Notably, in domains like material science and fluid dynamics, physical simulators are often used for evaluation of policy decisions (Fan et al., 2020; Rajak et al., 2021). These simulations tend to be computationally intensive. For example, reward calculation using flow simulation in Fan et al. (2020) requires about 1.1 hours for each episode. Other computationally demanding tasks include Neural Architecture Search (Zoph & Le, 2017) and Ordering-Based Causal Discovery (Wang et al., 2021). Furthermore, tasks like person re-identification (Liu et al., 2019) and RL with Human Feedback (RLHF) (Ouyang et al., 2022) demand human annotation for reward computation. This human annotation process often becomes a bottleneck, emphasizing the need to minimize such instances. While recent approaches like RL with AI feedback (RLAIF) (Lee et al., 2023) show promise, querying external AI typically incurs a cost, reinforcing the desire to minimize the number of reward evaluations.

Given this backdrop, our primary motivation is to alleviate the computational and monetary burdens of reward evaluations in the online RL setup. We try to accelerate policy gradient methods by employing a novel approach that efficiently selects a representative subset of episodes for reward computations.

Let us start with introducing necessary notations in RL and policy gradient methods.

1.1 MDP and policy

We are given a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and transition probability $p(\cdot|s, a)$ over $\mathcal{S}$ conditioned on each state-action pair along with a distribution over initial states $\rho(s)$; the tuple of these defines a Markov decision process (MDP). In the RL setting, it is typically the case that both the transition probabilities $p(\cdot|s, a)$ and the initial state distribution $\rho(s)$ are unknown to the agent, requiring the agent to learn an effective policy through interaction with the environment.

Stochastic policy $\pi(\cdot|s)$ is a probability density over $\mathcal{A}$ (with a canonical reference measure) conditioned on each $s \in \mathcal{S}$. Given p and π, we can generate a Markov chain that starts from a state $s_0 \in \mathcal{S}$ possibly drawn from a probability distribution associated with the MDP and continues as $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim p(\cdot|s_t, a_t)$ for $t \geq 0$. We call such a Markov chain an *episode* $e = (s_t, a_t)_{t\geq 0}$; it can be of finite length $T = T(e)$ due to some termination rule of MDP. Since p is fixed in our setting, we might abuse the notation as $e \sim \pi$ to represent the stochasticity of e.

Given an episode $e = (s_t, a_t)_{t\geq 0}$, the *discounted return* at time t is defined as $R_t(e) := \sum_{u\geq t} \gamma^{u-t} r(s_u, a_u)$, where $\gamma \in (0, 1)$ is a discount rate. When the dependency on e is apparent, we might simply denote $r(e) = (r_t)_{t\geq 0} = (r(s_t, a_t))_{t\geq 0}$ and $R_t = \sum_{u\geq t} \gamma^{u-t} r_u$. We finally define the Q-function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the value function $V^\pi : \mathcal{S} \to \mathbb{R}$ associated with the policy π by

$$Q^\pi(s, a) := \mathbb{E}_{e\sim\pi}[R_0(e)|s_0 = s, a_0 = a], \qquad V^\pi(s) := \mathbb{E}_{e\sim\pi}[R_0(e)|s_0 = s] = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s, a)].$$

1.2 Policy gradient

With a parameterized stochastic policy π_θ, we aim to maximize $J(\pi_\theta) := \mathbb{E}_{e\sim\pi_\theta}[R_0(e)]$ with respect to θ. Its gradient can be written as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{e\sim\pi}\left[\sum_{t\geq 0} \gamma^t R_t(e) \nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \mathbb{E}_{e\sim\pi}\left[\sum_{t\geq 0} \gamma^t A^\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)\right],$$

where $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$ is the advantage function, introduced here for variance reduction.

In practice, we usually approximate the gradient by Monte Carlo integration by generating episodes $e_1, \ldots, e_N \sim_{\text{iid}} \pi$:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \hat{G}(e_i), \tag{1}$$

$$\hat{G}(e) := \sum_{t\geq 0} \gamma^t \hat{A}_t(e) \nabla_\theta \log \pi_\theta(a_t|s_t), \tag{2}$$

where $\hat{A}_t(e)$ is an approximation of $A^\pi(s_t, a_t)$, *which requires evaluations of* $r(s_t, a_t)$ *for* $t \geq 0$. A typical choice for $\hat{A}$ is $\hat{A}_t(e) = R_t(e) - V_\varphi(s_t)$, where V_φ is a parametric approximation of the value function, often referred to as the baseline and updated iteratively (Williams, 1992). We refer to this method as the vanilla policy gradient (vpg).

We shall denote the policy parameter by θ and all the other parameters including baseline and kernel hyperparameters by φ. In order to highlight our motivation, we assume that we can separate *running an episode* e and *evaluating the reward* $r(e)$ (and so $\hat{A}_t(e)$); this policy gradient method is summarized in Algorithm 1.

Algorithm 1 Policy gradient

```
Input: A policy $\pi_\theta$, advantage estimator $\hat{A}$
for iteration $= 1, 2, \ldots$ do
    Generate episodes $(e_i)_{i=1}^N \sim_{\text{iid}} \pi_\theta$
    Compute $(r(e_i))_{i=1}^N$ and then $(\hat{A}(e_i))_{i=1}^N$
    $\theta \leftarrow \theta + \alpha N^{-1} \sum_{i=1}^N \hat{G}(e_i)$ ($\alpha$: learning rate)
    Update $\hat{A}$ by using $(e_i, r(e_i))_{i=1}^N$
end for
```

1.3 Contribution

To speed up the usual policy gradient methods, we propose combining policy gradient algorithms and kernel quadrature for reducing episodes, particularly aiming at expensive-reward situations mentioned in Section 1.

Kernel quadrature in this setting runs as follows: given a positive definite kernel of episodes $K(e, e')$, we approximate the N-point empirical measure of episodes by a weighted n-point subset: $\frac{1}{N}\sum_{i=1}^{N}\delta_{e_i} \approx \sum_{i\in I} w_i\delta_{e_i}$ with $I \subset \{1, \ldots, N\}$ and $|I| = n$. This process is treated as a (black-box) function $\mathrm{KQuad}(K, (e_i)_{i=1}^{N})$ given by Hayakawa et al. (2022). See Remark 3 for the computational complexity.

Algorithm 2 Vanilla PGKQ

```
Input: $n \ll N$, a policy $\pi_\theta$, advantage estimator $\hat{A}$, and episodic kernel $K$
for $iteration = 1, 2, \ldots$ do
    Generate episodes $(e_i)_{i=1}^{N} \sim_{\mathrm{iid}} \pi_\theta$
    $I, (w_i)_{i\in I} \leftarrow \mathrm{KQuad}(K, (e_i)_{i=1}^{N})$ with $|I| = n$
    Compute $(r(e_i))_{i\in I}$ and then $(\hat{A}(e_i), \hat{G}(e_i))_{i\in I}$
    $\theta \leftarrow \theta + \alpha \sum_{i\in I} w_i \hat{G}(e_i)$
    Update $\hat{A}$ by using $(w_i, e_i, r(e_i))_{i\in I}$
    Update $K$ by using $(w_i, e_i, r(e_i))_{i\in I}$
end for
```

The proposed algorithm is given in Algorithm 2 as *policy gradient with kernel quadrature* (PGKQ). While the detailed explanation of kernel quadrature under an MDP is described in Section 3, our contributions are summarized as follows:

- We develop a theory to make available the kernel quadrature over episodes, by introducing a Gaussian process modeling of returns or rewards, which can also be combined with policy gradient relatives including PPO (Schulman et al., 2017).
- We compare two versions of our PGKQ method with existing policy gradients with small and large batch sizes, and see the efficiency of PGKQ in catching up with the large-batch policy gradient only by small-batch reward observations.

2 Related literature

2.1 Bayesian quadrature for policy gradient

The most directly relevant to our study is the application of Bayesian quadrature (O'Hagan, 1991) for estimating policy gradient (BQPG; Ghavamzadeh & Engel, 2007; Ghavamzadeh et al., 2016; Akella et al., 2021). They model the Q-function Q^π by a Gaussian process (GP; see Section 3.1 for a formal definition), and estimate the policy gradient $\nabla_\theta J(\pi_\theta)$ as a posterior of a vector-valued GP based upon (noisy) observations of Q^π. Their main contribution from the viewpoint of gradient estimate is placing a better weight than the uniform weight in (1).

Although we also use GP modeling and weighted gradient estimate, our methods are different from these existing studies in the following two points.

Episodes reduction. As already mentioned in Section 1.3, our objective is to approximate a large batch of episodes by a smaller batch of weighted episodes in order to reduce the number of reward computations, while the variants of BQPG are on how to better estimate the policy gradient by a fixed batch of episodes.

Flexible kernel selection. As is always the case with the use of Bayesian quadrature, we need to know the exact values of some integrals associated with the covariance kernel of the GP (e.g., Eq. (16) in Ghavamzadeh & Engel, 2007) to execute BQPG. They need to use a specific class of kernel due to this limitation, while our method is valid for any choice of kernel because of the existence of a larger empirical measure that we want to approximate.

2.2 Numerical integration in data science

Estimating intractable integrals with a small number of integrand evaluations classically ranges from Monte Carlo (Metropolis & Ulam, 1949), cubature (Stroud, 1971) to QMC (Dick et al., 2013), while the recent literature also includes Bayesian/kernel quadrature (O'Hagan, 1991; Chen et al., 2010; Bach, 2017), recombination (Litterer & Lyons, 2012; Tchernychova, 2016), coresets (Bachem et al., 2017), determinantal point processes (DPPs; Bardenet et al., 2020), or dataset distillation (Wang et al., 2018). It is impossible to explain each method in detail, but they all agree on approximating a (probability) distribution, which is typically a continuous distribution or large discrete data, by a small (weighted) set (i.e., a discrete measure with small support).

Let us mention some existing applications of these methods towards better/faster gradient estimates in data science, stochastic gradient descents (SGDs; Robbins & Monro, 1951) in particular. DPPs have already been applied to acquire better mini-batches for gradient estimates in SGDs (Zhang et al., 2017; 2019; Bardenet et al., 2021), while there also is an application of recombination with the same motivation (Cosentino et al., 2020). Small-GAN (Sinha et al., 2020) uses coresets for a better minibatch selection when training generative adversarial networks (GANs; Goodfellow et al., 2020). Kernel quadrature, a method of seeking coreset based on kernel-based discrepancy, has also been applied to iterative updates in data science; minibatch selection in a warped Bayesian quadrature (Adachi et al., 2022) and Bayesian optimization (Adachi et al., 2023a;b), and dictionary compression in model-based RL (Chakraborty et al., 2023).

Our contribution compared with these studies is on the GP modeling specific to MDPs and the resulting application of kernel quadrature to suited for efficiently estimating the policy gradient.

3 Kernel quadrature for reducing episodes

3.1 Kernel quadrature in general

Kernel quadrature is a way of approximating a large or continuous distribution by a small discrete distribution. For a positive definite kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and a probability distribution μ, the aim of kernel quadrature is to find a good quadrature rule $\mu_n = (w_i, x_i)_{i=1}^n$ — a set of weights $w_i \in \mathbb{R}$ and points $x_i \in \mathcal{X}$ that makes the following the worst-case error small:

$$\mathrm{wce}(\mu_n; K, \mu) := \sup_{\|f\|_{\mathcal{H}} \le 1} |\mu_n(f) - \mu(f)|,$$

where $\mathcal{H}$ is the reproducing kernel Hilbert space (RKHS) associated with the kernel K, $\mu_n(f) := \sum_{i=1}^n w_i f(x_i)$, and $\mu(f) := \mathbb{E}_{x\sim\mu}[f(x)]$.

There are many algorithms for this problem, including but not limited to a greedy algorithm called herding (Chen et al., 2010; Huszár & Duvenaud, 2012; Bach et al., 2012; Tsuji et al., 2022), weighted sampling methods (Bach, 2017; Belhadji et al., 2019; 2020; Belhadji, 2021; Epperly & Moreno, 2023), a subsampling methods called thinning (Dwivedi & Mackey, 2021; 2022; Shetty et al., 2022); see Hayakawa et al. (2022, Table 1) for a comparison of these methods. We use the convex kernel quadrature (Hayakawa et al., 2022; 2023) for its empirical competence, but our method can incorporate any kernel quadrature method feasible with a general kernel and a discrete space.

The choice of the kernel K is essential for the applications of kernel quadrature (Briol et al., 2017), and we propose exploiting the covariance kernels of GPs of the discounted returns $R_t(e)$ or the reward function r in Section 3.2, in order to set an appropriate kernel over the space of episodes.

Formally, a real-valued GP on $\mathcal{X}$ is a distribution over the space of functions $\mathcal{X} \to \mathbb{R}$, associated with a mean function $m : \mathcal{X} \to \mathbb{R}$ and a covariance kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, denoted as $\mathcal{GP}(m, K)$. It satisfies that, for a random function $f \sim \mathcal{GP}(m, K)$ and any $x_1, \dots, x_m \in \mathcal{X}$, $(f(x_i))_{i=1}^m$ follows a normal distribution with the mean vector $(m(x_i))_{i=1}^m$ and the covariance matrix $(K(x_i, x_j))_{i,j=1}^m$. We write the expectation with respect to the distribution of the Gaussian process we have as $\mathbb{E}_{\mathcal{GP}}$ (e.g., $\mathbb{E}_{\mathcal{GP}}[f] = m$), and call $f \sim \mathcal{GP}(m, K)$ *centered* if $m = 0$ as a function.

Before going into details of the modeling in the case of MDP, we give an important relation between kernel quadrature and GP. The following is an immediate generalization of Huszár & Duvenaud (2012, Proposition 1):

Proposition 1. *Let $f \sim \mathcal{GP}(0, K)$ be a centered GP on $\mathcal{X}$ and μ be a probability distribution satisfying $\int_{\mathcal{X}} K(x,x)\,\mathrm{d}\mu(x) < \infty$. Then, for a kernel quadrature rule μ_n on the same space, we have*

$$\mathbb{E}_{\mathcal{GP}}[(\mu_n(f) - \mu(f))^2] = \mathrm{wce}(\mu_n; K, \mu)^2.$$

Note that the GP being centered is essential since otherwise an additional term of $(\mu_m(m) - \mu(m))^2$ with $m = \mathbb{E}_{\mathcal{GP}}[f]$ appears. We give its proof in Section A.1 for completeness. In the context of our methods, μ is typically given by a large batch of episodes and μ_n is a quadrature rule with a smaller support, with which we estimate the policy gradient. For more details on the role of this proposition, see the following sections, e.g., the description after Proposition 2.

3.2 Gaussian process modeling of MDP

Recall the notations for MDPs introduced in Section 1. In particular, we are given a parametric policy $\pi = \pi_\theta$, $\hat{A}_t(e)$ is an approximation of the advantage $A^\pi(s_t, a_t)$, and $\hat{G}_t(e)$ is an episodic gradient defined by (2). Also, recall that our objective is to find a good subset of episodes *before* evaluating rewards. Our use of GP is based on the following heuristic/informal assumption:

Assumption A (informal). *For a set of episodes $(e_i)_{i=1}^N$ and weights $(w_i)_{i\in I}$ with $I \subset \{1, \ldots, N\}$, if the uncertainty of $\frac{1}{N}\sum_{i=1}^N \sum_{t\ge 0} \gamma^t \hat{A}_t(e_i) - \sum_{i\in I} w_i \sum_{t\ge 0} \gamma^t \hat{A}_t(e_i)$ is small, then the uncertainty of $\frac{1}{N}\sum_{i=1}^N \hat{G}(e_i) - \sum_{i\in I} w_i \hat{G}(e_i)$ is also small.*

The *uncertainty* mentioned above is not a mathematical term, but can artificially be introduced by using a GP. By looking at (2), we are assuming above that ignoring the vector-valued term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ does not so much harm the quality of the reduced episodes. This heuristic not only simplifies the derivation of kernels into the MDP context, but also provides a unified treatment to similar policy-gradient methods such as TRPO and PPO (Schulman et al., 2015; 2017) where the loss functions are given by multiplying the advantage function and other score-related terms.

Provided the above heuristic, what we need to do is derive a GP for the following functional of an episode:

$$\hat{F}(e) := \sum_{t\ge 0} \gamma^t \hat{A}_t(e), \tag{3}$$

as we can then use kernel quadrature via Proposition 1. We want to directly model $\hat{F}$, but will start by modeling smaller components such as R_t and r for more data and flexibility.

In the following, we introduce two ways of modeling $\hat{A}$ with an episodic GP. These are based on a simpler *base GP* for either the return R_t or a reward r. Although we primarily consider the estimator $\hat{A}_t(e) = R_t(e) - V_\varphi(s_t)$, our argument can be generalized to more complicated estimators such as $R_t(e) - V_\varphi(s_t) - \gamma^{t_0 - t}(R_{t_0}(e) - V_\varphi(s_{t_0}))$ (for a certain t_0) introduced by Mnih et al. (2016).

For simplicity of notation, we shall write

$$z = (s, a),\ z_t = (s_t, a_t) \in \mathcal{Z} := \mathcal{S} \times \mathcal{A}$$

and $e = (s_t, a_t)_{t\ge 0} = (z_t)_{t\ge 0}$ for state-action pairs in the following. We present two ways of GP modeling in Sections 3.2.1 & 3.2.2, and we explain how to update the GPs (mean function, covariance kernel) over iterations in Section 3.2.3.

3.2.1 Option 1: modeling R_t with GP

In this model, our base GP on $\mathcal{Z}$ is given by

$$R_t|_{z_t = z} \sim \mathcal{GP}(V_\varphi, k_\psi), \tag{4}$$

where $V_\varphi(z) := V_\varphi(s)$ is the baseline function in the policy gradient algorithm and k_ψ is a positive definite kernel on the domain $\mathcal{Z} = \mathcal{S} \times \mathcal{A}$ with hyperparameter ψ. It formally means, for episodes $e = (z_t)_{t\geq 0}$, $e' = (z'_t)_{t\geq 0}$ and time t, u, we have ($\mathbb{C}$ov denotes covariance)

$$\mathbb{E}_{\mathcal{GP}}[R_t(e)] = V_\varphi(s_t), \tag{5}$$

$$\mathbb{C}\mathrm{ov}_{\mathcal{GP}}[R_t(e)R_u(e')] = k_\psi(z_t, z'_u). \tag{6}$$

Our intuition behind this modeling is two-fold. First, we $Q^\pi \sim \mathcal{GP}(V_\varphi, k_{0,\psi})$ to quantify the uncertainty of Q^π. Second, we consider R_t as an estimator of $Q^\pi(s_t, a_t)$, whose variance is modeled by another independent GP, i.e., $R_t|_{z_t=z} \sim \mathcal{GP}(Q^\pi(z), k_{1,\psi})$; this randomness is of a different kind from the above uncertainty, since R_t is still a random variable when fixing a stochastic policy π (or a stochastic environment), while Q^π is a deterministic function. The modeling (4) is obtained by combining these GPs ($k_\psi = k_{0,\psi} + k_{1,\psi}$).

Given the modeling for R_t, the advantage estimator $\hat{A}_t(e) = R_t(e) - V_\varphi(s_t)$ follows a centered GP of z_t (and then the episode e) with a covariance kernel k_ψ and we have the following kernel for $\hat{F}$. Similar computations also apply to other modeling such as $R_t(e) - V_\varphi(s_t) - \gamma^{t_0-t}(R_{t_0}(e) - V_\varphi(s_{t_0}))$.

Proposition 2. *Suppose* $(R_t)_{t\geq 0}$ *(as functions of an episode) follow a GP determined by* (5) *and* (6) *with* k_ψ *being bounded. Then, the functional*[1] $\hat{F}(e) = \sum_{t\geq 0} \gamma^t(\hat{A}_t(e))$ *with* $\hat{A}_t(e) = R_t(e) - V_\varphi(s_t)$ *follows a centered GP with a covariance kernel* K *given by*

$$K(e, e') = \sum_{t,u\geq 0} \gamma^{t+u} k_\psi(z_t, z'_u), \tag{7}$$

where $e = (z_t)_{t\geq 0}$, $e' = (z'_u)_{u\geq 0}$ *are episodes.*

Given $\hat{F} \sim \mathcal{GP}(0, K)$ with (7), we can apply Proposition 1 to see how kernel quadrature works. Indeed, by letting $f = \hat{F}$ and $\mu = \frac{1}{N}\sum_{i=1}^N \delta_{e_i}$ in Proposition 1, for a quadrature rule $\mu_n = (w_i, e_i)_{i\in I}$, we have

$$\mathbb{E}_{\mathcal{GP}}\left[\left(\frac{1}{N}\sum_{i=1}^N \hat{F}(e_i) - \sum_{i\in I} w_i \hat{F}(e_i)\right)^2\right] = \mathrm{wce}(\mu_n; K, \mu)^2.$$

Thus, provided Assumption A, a good kernel quadrature (i.e., that of a small worst-case error) gives us a good gradient estimate $\sum_{i\in I} w_i \hat{G}(e_i)$ while the amount of actual reward computations is kept small.

The actual algorithm looks like Algorithm 2, where we first update k_ψ (see Section 3.2.3) and then K as (7).

Remark 1. We could choose to use a parametric mean function $m_\psi(z)$ instead of using $V_\varphi(s)$ in (4) for more detailed modeling of R_t. However, then $\hat{F}$ could not represent the sum of advantage estimators and would not necessarily follow a centered GP, so we could not use Proposition 1. One way to circumvent this issue is using a hybrid gradient estimate described in the following section.

3.2.2 Option 2: modeling r with GP

The object modeled by the GP in the previous section is not static over the iterations of the policy gradient in that R_t (and Q^π) is dependent on the policy π. So the update of k_ψ might be inaccurate for future policies.

Our second option is thus modeling an static object, the reward function r:

$$r \sim \mathcal{GP}(m_\psi, k_\psi), \tag{8}$$

where ψ is a hyperparameter for the mean function m_ψ and the positive definite kernel k_ψ on $\mathcal{Z} = \mathcal{S} \times \mathcal{A}$.

However, under the modeling (8), the estimator $\hat{A}_t = R_t - V_\varphi(s_t)$ does not necessarily follow a centered GP because $\mathbb{E}_{\mathcal{GP}}[\hat{A}_t(e)] = \sum_{u\geq t} \gamma^{u-t} m_\psi(z_u) - V_\varphi(s_t)$. As already pointed out in Remark 1, we cannot simply

---

[1] Strictly speaking, we only justify that the sum in $\hat{F}$ converges in the $L^2(\mathcal{GP})$ space, when with infinite horizon; see Section A.2. The same applies to Proposition 3.

use Proposition 1 to run an episodic kernel quadrature in this case. Instead, we can observe the following decomposition:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{e\sim\pi}\left[\sum_{t\geq 0}\gamma^t(R_t - b(s_t))g_\theta(z_t)\right] \tag{9}$$

$$= \underbrace{\mathbb{E}_{e\sim\pi}\left[\sum_{t\geq 0}\gamma^t(R_t - \mathbb{E}_{\mathcal{GP}}[R_t])g_\theta(z_t)\right]}_{\text{I: centered GP term}} + \underbrace{\mathbb{E}_{e\sim\pi}\left[\sum_{t\geq 0}\gamma^t(\mathbb{E}_{\mathcal{GP}}[R_t] - b(s_t))g_\theta(z_t)\right]}_{\text{II: bias term}}, \tag{10}$$

where $b : \mathcal{S} \to \mathbb{R}$ is any (integrable) baseline function and $g_\theta(z_t) := \nabla_\theta \log \pi_\theta(a_t|s_t)$.

This decomposition can nicely be understood by introducing a *fake* reward m_ψ (instead of r) and the corresponding fake return $R_t^\psi(e) := \sum_{t\geq 0}\gamma^{u-t}m_\psi(z_u) = \mathbb{E}_{\mathcal{GP}}[R_t(e)]$. Indeed, the term II is given just by replacing R_t with R_t^ψ in the right-hand side of (9), and the remaining term I works as a gradient modification. Let us consider the context of reducing episodes from a large batch $(e_i)_{i=1}^N$ to a weighted small batch $\mu_n = (w_i, e_i)_{i\in I}$. Since we can compute the fake rewards/returns without access to r, we can estimate the bias term II by using all the episodes $(e_i)_{i=1}^N$.

For the term I, now that the integrands $R_t - \mathbb{E}_{\mathcal{GP}}[R_t]$ are centered, we can apply Proposition 1 with the following representation of the GP for the modified functional $\hat{F}_{\mathcal{GP}} := \sum_{t\geq 0}\gamma^t(R_t - \mathbb{E}_{\mathcal{GP}}[R_t])$.

Proposition 3. *Let the reward r follows a GP given by* (8) *with m_ψ and k_ψ being bounded. Then, the functional $\hat{F}_{\mathcal{GP}}(e) = \sum_{t\geq 0}\gamma^t(R_t(e) - \mathbb{E}_{\mathcal{GP}}[R_t(e)])$ follows a centered GP with a covariance kernel K given by*

$$K(e, e') = \sum_{t,u\geq 0}(1+t)(1+u)\gamma^{t+u}k_\psi(z_t, z'_u), \tag{11}$$

where $e = (z_t)_{t\geq 0}$, $e' = (z'_u)_{u\geq 0}$ are episodes.

Let us now define $\hat{F}_b := \sum_{t\geq 0}\gamma^t(R_t - b(s_t))$ as a generalization of $\hat{F}$ in the previous sections and $\hat{F}_b^\psi := \sum_{t\geq 0}\gamma^t(R_t^\psi - b(s_t))$ $(= \hat{F}_b - \hat{F}_{\mathcal{GP}})$ be its fake counterpart. Similarly to the previous section, given episodes $(e_i)_{i=1}^N$ and $\mu = \frac{1}{N}\sum_{i=1}^N \delta_{e_i}$, we can approximate $\frac{1}{N}\sum_{i=1}^N \hat{F}_b(e_i)$ by $\sum_{i\in I} w_i\hat{F}_{\mathcal{GP}}(e_i) + \frac{1}{N}\sum_{i=1}^N \hat{F}_b^\psi(e_i)$, where $\mu_n = (w_i, e_i)_{i\in I}$ is a quadrature rule, given by running a kernel quadrature algorithm only regarding the term I. Note that this approximation is computable only with the reward evaluations over episodes $(e_i)_{i\in I}$.

By using Proposition 1, their mean squared error in terms of GP, which is actually just about the error of $\frac{1}{N}\sum_{i=1}^N \hat{F}_{\mathcal{GP}}(e_i) - \sum_{i\in I} w_i\hat{F}_{\mathcal{GP}}(e_i)$, can again be represented as $\mathrm{wce}(\mu_n; K, \mu)^2$, where K is given by (11) regarding the term I this time.

However, the gradient estimate is not as simple as $\sum_{i\in I} w_i\hat{G}(e_i)$ in the previous section (or Assumption A), due to the use of a non-centered GP. Under the notation of (10), let us define

$$\hat{G}_b(e) := \sum_{t\geq 0}\gamma^t(R_t(e) - b(s_t))g_\theta(z_t),$$

$$\hat{G}_{\mathcal{GP}}(e) := \sum_{t\geq 0}\gamma^t(R_t(e) - \mathbb{E}_{\mathcal{GP}}[R_t])g_\theta(z_t),$$

$$\hat{G}_b^\psi(e) := \sum_{t\geq 0}\gamma^t(R_t^\psi(e) - b(s_t))g_\theta(z_t),$$

and then $\hat{G}_b^\psi$ $(= \hat{G}_b - \hat{G}_{\mathcal{GP}})$ is computable without access to the true rewards. By using these notations, we replace $\frac{1}{N}\sum_{i=1}^N \hat{G}_b(e_i)$ with a lighter gradient estimate $\sum_{i\in I} w_i\hat{G}_{\mathcal{GP}}(e_i) + \frac{1}{N}\sum_{i=1}^N \hat{G}_b^\psi(e_i)$. The overall algorithm is given by Algorithm 3. The way we update GP parameters is described in Section 3.2.3.

Algorithm 3 PGKQ with non-centered GP

```
Input: $n \ll N$, a policy $\pi_\theta$, baseline $b$, GP-mean $m_\psi$ and covariance $k_\psi$ modeling $r \sim \mathcal{GP}(m_\psi, k_\psi)$
for iteration $= 1, 2, \dots$ do
    Generate episodes $(e_i)_{i=1}^N \sim_{\text{iid}} \pi_\theta$
    Compute $K$ with (11)
    $I, (w_i)_{i \in I} \leftarrow \text{KQuad}(K, (e_i)_{i=1}^N)$ with $|I| = n$
    Compute $(r(e_i))_{i \in I}$ and then $(\hat{G}_{\mathcal{GP}}(e_i))_{i \in I}$
    $\theta \leftarrow \theta + \alpha(\sum_{i \in I} w_i \hat{G}_{\mathcal{GP}}(e_i) + \frac{1}{N} \sum_{i=1}^N \hat{G}_b^\psi(e_i))$
    Update $b$ by using $(e_i)_{i=1}^N$ and $m_\psi$
    Update $m_\psi, k_\psi$ by using $(w_i, e_i, r(e_i))_{i \in I}$
end for
```

Remark 2. We propose using $b = V_\varphi$ taken from the base policy gradient algorithm. We could update V_φ by using the weighted episodes $(w_i, e_i)_{i \in I}$ like the updates of m_ψ (see Section 3.2.3), but we actually propose updating V_φ based on fake rewards m_ψ, which clearly separates the roles of the policy gradient (PG) and kernel quadrature (KQ), since $\hat{G}_b^\psi$ is the usual PG-estimate given the fake rewards. Indeed, the KQ side then receives episodes $(e_i)_{i=1}^N$ and outputs a smaller batch of weighted episodes $(w_i, e_i)_{i \in I}$, while the PG side runs a usual PG with all the episodes $(e_i)_{i=1}^N$ associated with the fake reward m_ψ, with a modification of its gradient estimate by adding $\sum_{i \in I} w_i \hat{G}_{\mathcal{GP}}(e_i)$.

By following this formulation, we can not only use $\hat{G}_b$ (or $\hat{G}_b^\psi$) but also incorporate any gradient estimator (in the PG-side for the *fake-reward* MDP) in Algorithm 3. See also Appendix B.1 for more implementation details, including how we combine PGKQ with PPO.

3.2.3 Updating GP networks

Let us describe how we update k_ψ (and m_ψ) during the iterations of PGKQ.

Updating k_ψ. Although we should ideally update the GP based on the Bayes rule in a purely Bayesian setting, our GPs have to treat either a non-static target function (like R_t) or too many data points since each timestep in the MDP is a data point for the GP. Following the maximum likelihood estimation (MLE) and the deep kernel learning (Wilson et al., 2016) framework, we set the loss function for k_ψ as

$$L_{\text{ker}}(k_\psi) = \boldsymbol{y}^\top k_\psi(\boldsymbol{z}, \boldsymbol{z}) \boldsymbol{y} + \log\det k_\psi(\boldsymbol{z}, \boldsymbol{z}) \tag{12}$$

and conduct the gradient descent over the iterations, where $\boldsymbol{z}$ is given by arranging data points $z_t = (s_t, a_t)$ from all the episodes $e \in (e_i)_{i \in I}$, which are *after* the reduction of kernel quadrature, and $\boldsymbol{y}$ is the corresponding observed values, such as $R_t - V_\varphi(s_t)$ in Option 1 and $r(z_t) - m_\psi(z_t)$ in Option 2. In practice, we conduct the gradient descent by splitting $(\boldsymbol{z}, \boldsymbol{y})$ into minibatches to avoid computing the full $k_\psi(\boldsymbol{z}, \boldsymbol{z})$.

Updating m_ψ for non-centered GPs. We can either pass the optimization of m_ψ into the deep kernel learning by using the same loss in (12), or just independently learn m_ψ by least-squares as we do in practice. For the least-squares learning of m_ψ, we can make use of the quadrature measure μ_n to incorporate the episode weights as

$$L_{\text{mean}} = \mathbb{E}_{e \sim \mu_n}\left[\sum_{t \geq 0} (1+t)\gamma^t (r_t - m_\psi(z_t))^2\right], \tag{13}$$

where the discount factor $(1+t)\gamma^t$ comes from (11).

Remark 3 (Computational Complexity)**.** Let us consider the overall complexity of the kernel quadrature with kernel learning part in PGKQ. In both options, the computation of the single $K(e, e')$ requires T^2-times computation of k_ψ (T is the episode length), and so, by combining with the kernel quadrature algorithm in Hayakawa et al. (2022) (where we use all the episodes for Nyström's method), it takes

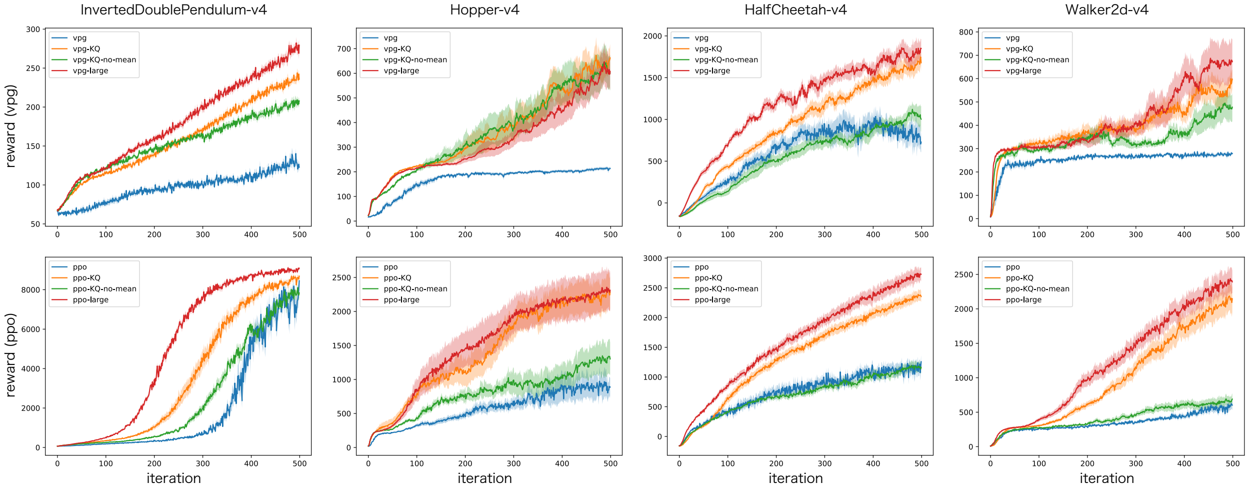

Figure 1: Learning curves in Mujoco tasks: reward vs iteration (comparing `{vpg, ppo}` (blue), `{vpg, ppo}-KQ` (orange), `{vpg, ppo}-KQ-no-mean` (green), and `{vpg, ppo}-large` (red))

$\mathcal{O}\big(N^2(\log n + T^2) + n^2 N \log(N/n)\big)$ to get an n-episode quadrature from an N-episode sample. If we set the batch size to M in the kernel learning, then the kernel update takes $\mathcal{O}\big(M^3\big)$ in each iteration and has $\mathcal{O}(nT/M)$ iterations. Thus, the overall complexity for a single iteration of the PGKQ algorithm (for kernel quadrature and kernel learning) is given by $\mathcal{O}\big(N^2(\log n + T^2) + n^2 N \log(N/n) + nTM^2\big)$.

4 Numerical experiments

To demonstrate the effectiveness of our proposed methods, we conducted experiments on MuJoCo tasks since they are widely recognized as standard benchmarks in RL, even though the reward calculation for them is lightweight.

We evaluated PGKQ using two options, both designed to reduce a large batch of N episodes to a smaller batch of n episodes. We adopted two conventional PG methods, vpg and PPO, as the bases for PGKQ. We also compared these methods without kernel quadrature, using either small n or large N batch sizes. Throughout this section, we maintained $N = 64$ and $n = 8$. Specifically, we compared the following algorithms:

- `{vpg, ppo}`: Corresponding algorithm with n episodes per iteration.
- `{vpg, ppo}-KQ`: Corresponding algorithm with PGKQ by modeling r (Option 2).
- `{vpg, ppo}-KQ-no-mean`: Corresponding algorithm with PGKQ by modeling R_t (Option 1).
- `{vpg, ppo}-large`: Corresponding algorithm with N episodes per iteration.

In each experiment (either based on `vpg` or `ppo`), for each of the four algorithms, we ran a 500-iteration optimization (whose one iteration is based on observing n or N episodes, depending on the algorithm) 10 times for statistical purposes. In all the figures, the empirical average of the total reward $\sum_{t\geq 0} r(s_t, a_t)$ (which is actually a finite sum in all the experiments) with its standard error (shaded region) are shown. All the experiments were conducted by using PyTorch (Paszke et al., 2019) and Adam (Kingma & Ba, 2015).

Our motivation for conducting these experiments is as follows: *How well can we catch up with the learning curve of* `{vpg, ppo}-large` *by using PGKQ rather than the small-batch* `{vpg, ppo}`*?* So we compared the rewards against iterations in this section, but we also compare rewards against steps in Appendix B.2.

In all the experiments, we used three-layer fully connected ReLU neural networks (NNs) for each of m_ψ and k_ψ, where $k_\psi(z, z')$ was computed by passing the NN-embeddings of state-action pairs z and z' to the Gaussian kernel with additional scale and noise parameters. See Appendix B.1 for implementation details.

MuJoCo tasks. We used MuJoCo (v2.1.0, Todorov et al., 2012) with the Gymnasium API (Towers et al., 2023)[2]. We compared all the aforementioned algorithms in the four following tasks: `InvertedDoublePendulum-v4`, `Hopper-v4`, `HalfCheetah-v4`, `Walker2d-v4`. In all the tasks, the maximum episode length is 1000, where the tasks except `HalfCheetah-v4` can terminate earlier when the state enters a predefined *unhealthy* region.

In these tasks, for the `vpg` (vanilla policy gradient) and `ppo` (proximal policy optimization with clipping, Schulman et al., 2017), we used the implementation of the machina[3] library. The learning rates of the policy, baseline, and GP-related networks were all set to 3×10^{-4}. This value was the default value of the machina library, but also chosen as the most successful learning rate when comparing several different learning rates with the 300-iteration, n-episode `ppo` in the `Hopper-v4` task. The discount rate was $\gamma = 0.995$.

The results are shown in Figure 1. In most tasks and base algorithms `{vpg, ppo}`, the order of total reward followed `{vpg, ppo}-large` > `{vpg, ppo}-KQ` > `{vpg, ppo}-KQ-no-mean` > `{vpg, ppo}`. In particular, in the experiments with `vpg`, PGKQ methods often get out of the plateau where the `vpg` is stuck.

5 Concluding remarks

We have developed a method called PGKQ, which combines existing policy gradient algorithm with kernel quadrature over episodes, aiming at better gradient estimates while keeping the number of actual reward computations small. PGKQ is based on GP models of the return R_t (Option 1) or the reward function r (Option 2), whose covariance kernel accumulated over episodes gives a kernel over the space of episodes, which allows the use of kernel quadrature for selecting a small but informative subset from a large batch of episodes.

We also confirmed the competitiveness of PGKQ, especially the Option 2, by performing experiments in MuJoCo tasks.

As a future direction, it should be beneficial to study a generalization of PGKQ based on vector-valued GPs as in existing BQPG approaches (Ghavamzadeh & Engel, 2007; Ghavamzadeh et al., 2016; Akella et al., 2021) to avoid the heuristic Assumption A.

While we have been focusing on online RL, extending our approach to offline RL represents another interesting direction. In such a setup, determining which samples from a large batch are essential for reward computation, given the current learning policy, becomes critically important.

References

Masaki Adachi, Satoshi Hayakawa, Martin Jørgensen, Harald Oberhauser, and Michael A Osborne. Fast Bayesian inference with batch Bayesian quadrature via kernel recombination. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16533–16547, 2022.

Masaki Adachi, Satoshi Hayakawa, Saad Hamid, Martin Jørgensen, Harald Oberhauser, and Micheal A Osborne. SOBER: Highly parallel Bayesian optimization and bayesian quadrature over discrete and mixed spaces. *arXiv preprint arXiv:2301.11832*, 2023a.

Masaki Adachi, Satoshi Hayakawa, Xingchen Wan, Martin Jørgensen, Harald Oberhauser, and Michael A Osborne. Domain-agnostic batch Bayesian optimization with diverse constraints via Bayesian quadrature. *arXiv preprint arXiv:2306.05843*, 2023b.

Ravi Tej Akella, Kamyar Azizzadenesheli, Mohammad Ghavamzadeh, Animashree Anandkumar, and Yisong Yue. Deep Bayesian quadrature policy optimization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6600–6608, 2021.

---

[2] All the experiments with MuJoCo were conducted with a Google Cloud Vertex AI notebook with an NVIDIA T4 (16-core vCPU, 60 GB RAM).

[3] `https://github.com/DeepX-inc/machina`

Francis Bach. On the equivalence between kernel quadrature rules and random feature expansions. *The Journal of Machine Learning Research*, 18(1):714–751, 2017.

Francis Bach, Simon Lacoste-Julien, and Guillaume Obozinski. On the equivalence between herding and conditional gradient algorithms. In *International Conference on Machine Learning*, pp. 1355–1362, 2012.

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Rémi Bardenet, Adrien Hardy, et al. Monte carlo with determinantal point processes. *The Annals of Applied Probability*, 30(1):368–417, 2020.

Rémi Bardenet, Subhroshekhar Ghosh, and Meixia Lin. Determinantal point processes based on orthogonal polynomials for sampling minibatches in SGD. In *Advances in Neural Information Processing Systems*, volume 34, pp. 16226–16237, 2021.

Ayoub Belhadji. An analysis of Ermakov–Zolotukhin quadrature using kernels. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Ayoub Belhadji, R Bardenet, and Pierre Chainais. Kernel quadrature with DPPs. In *Advances in Neural Information Processing Systems*, volume 32, pp. 12907–12917, 2019.

Ayoub Belhadji, Rémi Bardenet, and Pierre Chainais. Kernel interpolation with continuous volume sampling. In *International Conference on Machine Learning*, pp. 725–735. PMLR, 2020.

François-Xavier Briol, Chris J Oates, Jon Cockayne, Wilson Ye Chen, and Mark Girolami. On the sampling problem for kernel quadrature. In *International Conference on Machine Learning*, pp. 586–595. PMLR, 2017.

Souradip Chakraborty, Amrit Singh Bedi, Pratap Tokekar, Alec Koppel, Brian Sadler, Furong Huang, and Dinesh Manocha. Posterior coreset construction with kernelized stein discrepancy for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6980–6988, 2023.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *Conference on Uncertainty in Artificial Intelligence*, pp. 109–116, 2010.

Francesco Cosentino, Harald Oberhauser, and Alessandro Abate. Carathéodory sampling for stochastic gradient descent. *arXiv preprint arXiv:2006.01819*, 2020.

Josef Dick, Frances Y. Kuo, and Ian H. Sloan. High-dimensional integration: The quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

Raaz Dwivedi and Lester Mackey. Kernel thinning. In *Conference on Learning Theory*, pp. 1753–1753. PMLR, 2021.

Raaz Dwivedi and Lester Mackey. Generalized kernel thinning. In *International Conference on Learning Representations*, 2022.

Ethan N Epperly and Elvira Moreno. Kernel quadrature with randomly pivoted cholesky. *arXiv preprint arXiv:2306.03955*, 2023.

Dixia Fan, Liu Yang, Zhicheng Wang, Michael S. Triantafyllou, and George Em Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences of the United States of America*, 117(42), 2020.

Mohammad Ghavamzadeh and Yaakov Engel. Bayesian actor-critic algorithms. In *Proceedings of the 24th International Conference on Machine Learning*, pp. 297–304, 2007.

Mohammad Ghavamzadeh, Yaakov Engel, and Michal Valko. Bayesian policy gradient and actor-critic algorithms. *The Journal of Machine Learning Research*, 17(1):2319–2371, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144, 2020.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. In B. Schölkopf, J. Platt, and T. Hoffman (eds.), *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.

Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990.

Satoshi Hayakawa, Harald Oberhauser, and Terry Lyons. Positively weighted kernel quadrature via subsampling. In *Advances in Neural Information Processing Systems*, volume 35, pp. 6886–6900, 2022.

Satoshi Hayakawa, Harald Oberhauser, and Terry Lyons. Sampling-based Nyström approximation and kernel quadrature. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 12678–12699, 2023.

Ferenc Huszár and David Duvenaud. Optimally-weighted herding is Bayesian quadrature. In *Conference on Uncertainty in Artificial Intelligence*, pp. 377–386, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. RLAIF: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

C. Litterer and T. Lyons. High order recombination and an application to cubature on Wiener space. *The Annals of Applied Probability*, 22(4):1301–1327, 2012.

Zimo Liu, Jingya Wang, Shaogang Gong, Huchuan Lu, and Dacheng Tao. Deep reinforcement active learning for human-in-the-loop person re-identification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp. 1928–1937. PMLR, 2016.

Anthony O'Hagan. Bayes–Hermite quadrature. *Journal of Statistical Planning and Inference*, 29(3):245–260, 1991.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, J. Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, P. Welinder, P. Christiano, J. Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

Pankaj Rajak, Aravind Krishnamoorthy, Ankit Mishra, Rajiv Kalia, Aiichiro Nakano, and Priya Vashishta. Autonomous reinforcement learning agent for chemical vapor deposition synthesis of quantum materials. *npj Computational Materials*, 7(1), 2021.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Abhishek Shetty, Raaz Dwivedi, and Lester Mackey. Distribution compression in near-linear time. In *International Conference on Learning Representations*, 2022.

Samarth Sinha, Han Zhang, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, and Augustus Odena. Small-GAN: Speeding up GAN training using core-sets. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 9005–9015. PMLR, 2020.

Bharath K Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert RG Lanckriet. Hilbert space embeddings and metrics on probability measures. *The Journal of Machine Learning Research*, 11:1517–1561, 2010.

Arthur H Stroud. *Approximate calculation of multiple integrals.* Prentice-Hall, 1971.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Maria Tchernychova. *Carathéodory cubature measures.* PhD thesis, University of Oxford, 2016.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL `https://zenodo.org/record/8127025`.

Kazuma Tsuji, Ken'ichiro Tanaka, and Sebastian Pokutta. Pairwise conditional gradients without swap steps and sparser kernel herding. In *International Conference on Machine Learning*, pp. 21864–21883. PMLR, 2022.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Xiaoqiang Wang, Yali Du, Shengyu Zhu, Liangjun Ke, Zhitang Chen, Jianye Hao, and Jun Wang. Ordering-based causal discovery with reinforcement learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 3566–3573. International Joint Conferences on Artificial Intelligence Organization, 2021.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In Arthur Gretton and Christian C. Robert (eds.), *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51, pp. 370–378. PMLR, 09–11 May 2016.

Yuan Xia, Jingbo Zhou, Zhenhui Shi, Chao Lu, and Haifeng Huang. Generative adversarial regularized mutual information policy gradient framework for automatic diagnosis. In *AAAI Conference on Artificial Intelligence*, pp. 1062–1069, 2020.

Cheng Zhang, Hedvig Kjellström, and Stephan Mandt. Determinantal point processes for mini-batch diversification. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence*, 2017.

Cheng Zhang, Cengiz Öztireli, Stephan Mandt, and Giampiero Salvi. Active mini-batch sampling using repulsive point processes. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 33, pp. 5741–5748, 2019.

Barret Zoph and QuocQuoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A Proofs

A.1 Proof of Proposition 1

Proof. Suppose $f \sim \mathcal{GP}(0, K)$. We first compute the following term:

$$\mathbb{E}_{\mathcal{GP}}[\mu(f)^2] = \mathbb{E}_{\mathcal{GP}}\left[\iint f(x)f(y)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y)\right].$$

By Cauchy-Schwarz, we have $\iint |f(x)f(y)|\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y) \le (\int f(x)^2\,\mathrm{d}\mu(x))^{1/2}(\int f(y)^2\,\mathrm{d}\mu(y))^{1/2} = \int f(x)^2\,\mathrm{d}\mu(x)$, so we have

$$\mathbb{E}_{\mathcal{GP}}\left[\iint |f(x)f(y)|\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y)\right] \le \mathbb{E}_{\mathcal{GP}}\left[\int f(x)^2\,\mathrm{d}\mu(x)\right] = \int \mathbb{E}_{\mathcal{GP}}\left[f(x)^2\right]\,\mathrm{d}\mu(x) = \int K(x,x)\,\mathrm{d}\mu(x) < \infty,$$

where we have used Fubini's theorem (thanks to the nonnegativity of the integrand) in the first equality, and the assumption on the kernel in the last inequality. Thus, we can use Fubini's theorem for the original multiple integral, and we have, by using $\mathbb{E}_{\mathcal{GP}}[f(x)f(y)] = K(x, y)$,

$$\mathbb{E}_{\mathcal{GP}}\left[\mu(f)^2\right] = \iint \mathbb{E}_{\mathcal{GP}}[f(x)f(y)]\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y) = \iint K(x,y)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y).$$

From the same use of Fubini's theorem by (partially) replacing μ with μ_n, we can compute the other quadratic terms and obtain

$$\mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f) - \mu(f))^2\right] = \iint K(x,y)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu(y) - 2\sum_{i=1}^n w_i \int K(x_i, y)\,\mathrm{d}\mu(y) + \sum_{i,j=1}^n w_i w_j K(x_i, x_j),$$

which is a well-known formula for the $\mathrm{wce}(\mu_n; K, \mu)^2$ (Gretton et al., 2006; Sriperumbudur et al., 2010).

In general, we can also treat the non-centered case $f \sim \mathcal{GP}(m, K)$ if m is integrable with respect to μ. Indeed, we can apply the above computation to $f - m \sim \mathcal{GP}(0, K)$ to obtain

$$\mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f-m) - \mu(f-m))^2\right] = \mathrm{wce}(\mu_n; K, \mu)^2.$$

From the linearity of the integral, we actually have

$$\begin{aligned}
&\mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f-m) - \mu(f-m))^2\right] \\
&= \mathbb{E}_{\mathcal{GP}}\left[((\mu_n(f) - \mu(f)) - (\mu_n(m) - \mu(m)))^2\right] \\
&= \mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f) - \mu(f))^2\right] - 2(\mu_n(m) - \mu(m))\underbrace{\mathbb{E}_{\mathcal{GP}}[\mu_n(f) - \mu(f)]}_{=\mu_n(m)-\mu(m)} + (\mu_n(m) - \mu(m))^2 \\
&= \mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f) - \mu(f))^2\right] - (\mu_n(m) - \mu(m))^2.
\end{aligned}$$

Thus, in general, we have

$$\mathbb{E}_{\mathcal{GP}}\left[(\mu_n(f) - \mu(f))^2\right] = \mathrm{wce}(\mu_n; K, \mu)^2 + (\mu_n - \mu(m))^2.$$

□

A.2 Proof of Proposition 2

Proof. From the modeling (5) and (6), $\hat{A}_t$ formally follows a GP on the variables $(e,t) \in \mathcal{E} \times \mathcal{T}$ such that

$$\mathbb{E}_{\mathcal{GP}}\left[\hat{A}_t(e)\right] = 0, \qquad \mathbb{E}_{\mathcal{GP}}\left[\hat{A}_t(e)\hat{A}_u(e')\right] = k_\psi(z_t, z'_u),$$

where $\mathcal{E}$ is the space of episodes and $\mathcal{T} := \{0,1,2,\dots\}$ is the space of time indices.

Let us define the space $L^2(\mathcal{GP})$ as the space of $\mathbb{R}$-valued centered Gaussian variables given by taking the completion of the linear space $\operatorname{span}\{\hat{A}_t(e) \mid (e,t) \in \mathcal{E} \times \mathcal{T}\}$ with respect to the norm $\|X\|_{L^2(\mathcal{GP})} := \mathbb{E}_{\mathcal{GP}}\left[X^2\right]^{1/2}$. Then, since the kernel is bounded, we have $\sum_{t\ge 0}\|\gamma^t \hat{A}_t(e)\|_{L^2(\mathcal{GP})} = \sum_{t\ge 0}\gamma^t k(z_t,z_t)^{1/2} < \infty$, and so the infinite sum $\hat{F}(e) = \sum_{t\ge 0}\gamma^t \hat{A}_t(e)$ is well-defined in $L^2(\mathcal{GP})$. By letting $\hat{F}_T(e) := \sum_{t\ge 0}^T \gamma^t \hat{A}_t(e)$, we have $\hat{F}_T \to \hat{F}$ in $L^2(\mathcal{GP})$.

Therefore, we have that $\{\hat{F}(e) \mid e \in \mathcal{E}\}$ is a family of centered (jointly) Gaussian variables with

$$\begin{aligned}\mathbb{E}_{\mathcal{GP}}\left[\hat{F}(e)\hat{F}(e')\right] = \lim_{T\to\infty}\lim_{U\to\infty}\mathbb{E}_{\mathcal{GP}}\left[\hat{F}_T(e)\hat{F}_U(e')\right] &= \lim_{T\to\infty}\lim_{U\to\infty}\sum_{t=0}^T\sum_{u=0}^U \gamma^{t+u}\mathbb{E}_{\mathcal{GP}}\left[\hat{A}_t(e)\hat{A}_u(e')\right] \\ &= \lim_{T\to\infty}\lim_{U\to\infty}\sum_{t=0}^T\sum_{u=0}^U \gamma^{t+u}k_\psi(z_t,z'_u), \qquad (14)\end{aligned}$$

where $e = (z_t)_{t\ge 0}$, $e' = (z'_u)_{u\ge 0}$ are episodes. Since the kernel is bounded, the sum $\sum_{t,u\ge 0}\gamma^{t+u}k_\psi(z_t,z'_u)$ is absolutely convergent, and coincides with the right-hand side of (14), which completes the proof. ☐

A.3 Proof of Proposition 3

Proof. The flow of the proof is mostly the same as the previous one. The base GP is now $r \sim \mathcal{GP}(m_\psi, k_\psi)$. This time we define $L^2(\mathcal{GP})$ as the space of $\mathbb{R}$-valued Gaussian variable given by the completion of the linear space $\operatorname{span}(\{1\} \cup \{r(z) \mid z \in \mathcal{Z}\})$ with respect to the norm $\|X\|_{L^2(\mathcal{GP})} := \mathbb{E}_{\mathcal{GP}}\left[X^2\right]^{1/2}$. Since m_ψ and k_ψ are bounded, we first have that $R_t(e) = \sum_{u\ge t}\gamma^{u-t}r(z_u)$ is well-defined in $L^2(\mathcal{GP})$ as

$$\sum_{u\ge t}\|\gamma^{u-t}r(z_u)\|_{L^2(\mathcal{GP})} = \sum_{u\ge t}\gamma^{u-t}\sqrt{m(z_u)^2 + k_\psi(z_u,z_u)} < \infty.$$

Note that $L^2(\mathcal{GP})$ is a Hilbert space containing 1 with the inner product $\langle X, X'\rangle_{L^2(\mathcal{GP})} = \mathbb{E}_{\mathcal{GP}}[XX']$. Thus, the expectation, which is the inner product with 1, is continuous with respect to the norm, and so we have

$$\mathbb{E}_{\mathcal{GP}}[R_t(e)] = \lim_{T\to\infty}\mathbb{E}_{\mathcal{GP}}\left[\sum_{u\ge t}\gamma^{u-t}r(z_u)\right] = \lim_{T\to\infty}\sum_{u=t}^T\gamma^{u-t}m_\psi(z_u) = \sum_{u\ge t}\gamma^{u-t}m_\psi(z_u),$$

where the last infinite sum is absolutely convergent thanks to the boundedness of m_ψ. We can also prove that $R_t(e) - \mathbb{E}_{\mathcal{GP}}[R_t(e)] = \sum_{u\ge t}\gamma^{u-t}(r(z_u) - m_\psi(z_u))$ by combining two convergent sequences.

By letting $\hat{A}_t^{\mathcal{GP}}(e) := R_t(e) - \mathbb{E}_{\mathcal{GP}}[R_t(e)]$, we have a family of centered jointly Gaussian variables $\{\hat{A}_t^{\mathcal{GP}}(e) \mid (e,t) \in \mathcal{E}\times\mathcal{T}\}$ such that

$$\|\hat{A}_t^{\mathcal{GP}}(e)\|_{L^2(\mathcal{GP})} \le \sum_{u\ge t}\|\gamma^{u-t}(r(z_u) - m_\psi(z_u))\|_{L^2(\mathcal{GP})} \le \sum_{u\ge t}\gamma^{u-t}\sqrt{k_\psi(z_u,z_u)} < C$$

for a constant $C > 0$ independent of t and e, where $\mathcal{E}$ and $\mathcal{T}$ are the spaces of episodes and time indices as defined in the previous proof. Thus, $\hat{F}_{\mathcal{GP}}(e) = \sum_{t\ge 0}\gamma^t\hat{A}_t^{\mathcal{GP}}(e)$ is well-defined in $L^2(\mathcal{GP})$ and $\{\hat{F}_{\mathcal{GP}}(e) \mid e \in \mathcal{E}\}$ is a family of centered jointly Gaussian variables. Since the double sum $\sum_{t\ge 0}\gamma^t\sum_{u\ge t}\|\gamma^{u-t}(r(z_u) - m_\psi(z_u))\|_{L^2(\mathcal{GP})}$ is absolutely convergent, we can exchange the sum as

$$\hat{F}_{\mathcal{GP}}(e) = \sum_{t\ge 0}\gamma^t\sum_{u\ge t}\gamma^{u-t}(r(z_u) - m_\psi(z_u)) = \sum_{u\ge 0}\sum_{u=0}^t\gamma^u(r(z_u) - m_\psi(z_u)) = \sum_{u\ge 0}(1+u)\gamma^u(r(z_u) - m_\psi(z_u)).$$

Therefore, the covariance kernel for $\hat{F}_{\mathcal{GP}}(e)$ can formally be computed as

$$\begin{aligned}\mathbb{E}_{\mathcal{GP}}\left[\hat{F}_{\mathcal{GP}}(e)\hat{F}_{\mathcal{GP}}(e')\right] &= \sum_{t,u\ge 0}(1+t)(1+u)\gamma^{t+u}\mathbb{E}_{\mathcal{GP}}[(r(z_t)-m_\psi(z_t))(r(z'_u)-m_\psi(z'_u))]\\ &= \sum_{t,u\ge 0}(1+t)(1+u)\gamma^{t+u}k_\psi(z_t,z'_u)\end{aligned}$$

for episodes $e=(z_t)_{t\ge 0}$, $e'=(z'_u)_{u\ge 0}$, which is justified by the same logic as in the previous proof. □

B Experimental details

B.1 Implementation details

B.1.1 Combining policy gradient with kernel quadrature

Recall that we have introduced in (2) the single-episode gradient estimate

$$\hat{G}(e)=\sum_{t\ge 0}\gamma^t\hat{A}_t(e)\nabla_\theta\log\pi_\theta(a_t|s_t) \tag{15}$$

with an advantage estimator $\hat{A}_t$. Although we have written that we use $\frac{1}{N}\sum_{i=1}^N\hat{G}(e_i)$ as the Monte Carlo gradient estimate, what we write in the actual code is the computation of the (one-dimensional) loss

$$(\text{Loss})=\frac{1}{N}\sum_{i=1}^N L_{\text{vpg}}[\hat{A}_t](e_i),\quad\text{where}\quad L_{\text{vpg}}[\hat{A}_t](e):=\sum_{t\ge 0}\gamma^t\hat{A}_t(e)\log\pi_\theta(a_t|s_t),$$

and running an automatic differentiation to get its gradient with respect to the parameter θ. Here, $\hat{A}_t$ is treated as just a functional of an episode in the definition of $L_{\text{vpg}}[\hat{A}_t]$. Let us explain how we actually compute the policy loss in the PGKQ given a kernel quadrature rule $\mu_n=(w_i,e_i)_{i\in I}$. We start from VPG (vanilla policy gradient).

(v1) VPG with kernel quadrature with a centered GP ($\mathbb{E}_{\mathcal{GP}}[\hat{A}_t]=0$, Option 1) is the easiest case. Given μ_n, we just replace $\frac{1}{N}\sum_{i=1}^N L_{\text{vpg}}[\hat{A}_t](e_i)$ with $\sum_{i\in I}w_iL_{\text{vpg}}[\hat{A}_t](e_i)$ in loss computation.

(v2) When we combine VPG and a non-centered GP modeling of r (Option 2), we just exploit the decomposition (10) for loss computation (with a baseline function b):

$$(\text{Loss})=\sum_{i\in I}w_iL_{\text{vI}}(e_i)+\frac{1}{N}\sum_{i=1}^N L_{\text{vII}}(e_i),$$

$$\text{where}\quad\begin{cases}L_{\text{vI}}(e):=\sum_{t\ge 0}\gamma^t(R_t(e)-\mathbb{E}_{\mathcal{GP}}[R_t(e)])\log\pi_\theta(a_t|s_t),\\ L_{\text{vII}}(e):=\sum_{t\ge 0}\gamma^t(\mathbb{E}_{\mathcal{GP}}[R_t(e)]-b(s_t))\log\pi_\theta(a_t|s_t).\end{cases}$$

As the baseline, we use the value estimator V_φ trained with fake rewards as explained in Remark 2. The observation $L_{\text{vI}}=L_{\text{vpg}}[R_t-\mathbb{E}_{\mathcal{GP}}[R_t]]$ and $L_{\text{vII}}=L_{\text{vpg}}[\mathbb{E}_{\mathcal{GP}}[R_t]-b]$ allows a unified implementation.

We can also apply PGKQ to PPO (Schulman et al., 2017). In the usual PPO, we use the probability ratio (as a functional of an episode) $q_t^\theta(e):=\pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$, where θ_{old} is the policy parameter at which we assume the episodes $e_1,\dots,e_N$ have been drawn. Given an advantage estimator $\hat{A}_t$ as a functional of an episode, we compute the loss as follows:

$$(\text{Loss})=\frac{1}{N}\sum_{i=1}^N L_{\text{ppo}}[\hat{A}_t](e_i),\quad\text{where}\quad L_{\text{ppo}}[\hat{A}_t](e):=\sum_{t\ge 0}\gamma^t\min\{q_t^\theta(e)\hat{A}_t(e),\ \text{clip}(q_t^\theta(e),1-\varepsilon,1+\varepsilon)\hat{A}_t(e)\}.$$

Here, $\text{clip}(a,b,c):=\min\{\max\{a,b\},c\}$ and ε (= 0.2 in the implementation) is a clipping parameter.

(p1) PPO with the centered GP modeling (Option 1) is given by a straightforward replacement of L_{vpg} by L_{ppo}

(p2) When combining PPO with the Option 2, we can also imitate the decomposition of (v2):

$$(\text{Loss}) = \sum_{i \in I} w_i L_{\text{pI}}(e_i) + \frac{1}{N} \sum_{i=1}^{N} L_{\text{pII}}(e_i), \quad \text{where} \quad L_{\text{pI}} := L_{\text{ppo}}[R_t - \mathbb{E}_{\mathcal{GP}}[R_t]],\ L_{\text{pII}} := L_{\text{ppo}}[\mathbb{E}_{\mathcal{GP}}[R_t] - b].$$

Here, b is again the value estimator V_φ trained by fake rewards.

We can also consider the use of other advantage estimators than $R_t - b$. Indeed, $\hat{A}'_t(e) = R_t(e) - V_\varphi(s_t) - \gamma^{t_0 - t}(R_{t_0}(e) - V_\varphi(s_{t_0}))$ with t_0 being the final time was adopted in the MuJoCo tasks (also following the original code of Machina). The modification of (v1) and (p1) is straightforward as they formally do not depend on the specific form of $\hat{A}_t$. For (v2) and (p2), though there are other possibilities, we just replaced L_{vII} and L_{pII} with $L'_{\text{vII}} := L_{\text{vpg}}[\mathbb{E}_{\mathcal{GP}}[\hat{A}'_t]]$ and $L'_{\text{pII}} := L_{\text{ppo}}[\mathbb{E}_{\mathcal{GP}}[\hat{A}'_t]]$, where $\mathbb{E}_{\mathcal{GP}}[\hat{A}'_t](e) := \mathbb{E}_{\mathcal{GP}}[R_t(e)] - V_\varphi(s_t) - \gamma^{t_0 - t}(\mathbb{E}_{\mathcal{GP}}[R_{t_0}(e)] - V_\varphi(s_{t_0}))$ is regarded as a functional of an episode.

B.1.2 Network architecture

For the policy and baseline networks, we used the default implementations the library (Machina). Let us explain the part with our original implementation, m_ψ and k_ψ in the GP modeling.

Inputs. In all the MuJoCo tasks, we simply concatenated two vectors s_t and a_t to make z_t, and used it as inputs to our networks. Let us denote by D the dimension of z_t in the following.

Implementation of k_ψ. In both experiments, we implemented k_ψ as follows

$$k_\psi(z_t, z'_u) = \exp\left(\lambda_\psi - \frac{\|f_\psi(z_t) - f_\psi(z'_u)\|^2}{20}\right) + (10^{-5} + \exp(\sigma_\psi))\delta(z_t, z'_u),$$

where $\|\cdot\|$ is the Euclidean norm, $\delta(z_t, z'_u)$ equals 1 if $z_t = z'_u$ and 0 otherwise, $\lambda_\psi, \sigma_\psi \in \mathbb{R}$ are respectively the scale and noise parameters, and $f_\psi : \mathbb{R}^D \to \mathbb{R}^{10}$ is the embedding function explained in the following.

For f_ψ in all the MuJoCo tasks, we used a fully connected neural network with two hidden layers with D (so $\mathbb{R}^D \to \mathbb{R}^D \to \mathbb{R}^D \to \mathbb{R}^{10}$) and the ReLU activations except for the final layer.

Implementation of m_ψ. We used a fully connected neural network with two hidden layers with dimensions 200 and 100 (so $\mathbb{R}^D \to \mathbb{R}^{200} \to \mathbb{R}^{100} \to \mathbb{R}^{10}$) and the ReLU activations except for the final layer.

B.2 Reward vs step

Though we only compared the rewards against iterations in the main body, since it is common to compare the rewards against observed steps (e.g., Schulman et al., 2017), we here present our experimental results in that regard. Figure 2 is on each of the four MuJoCo tasks corresponding to Figure 1. Note that this is not necessarily a fair comparison to `{vpg,ppo}-large` in terms of the learning rate per step.

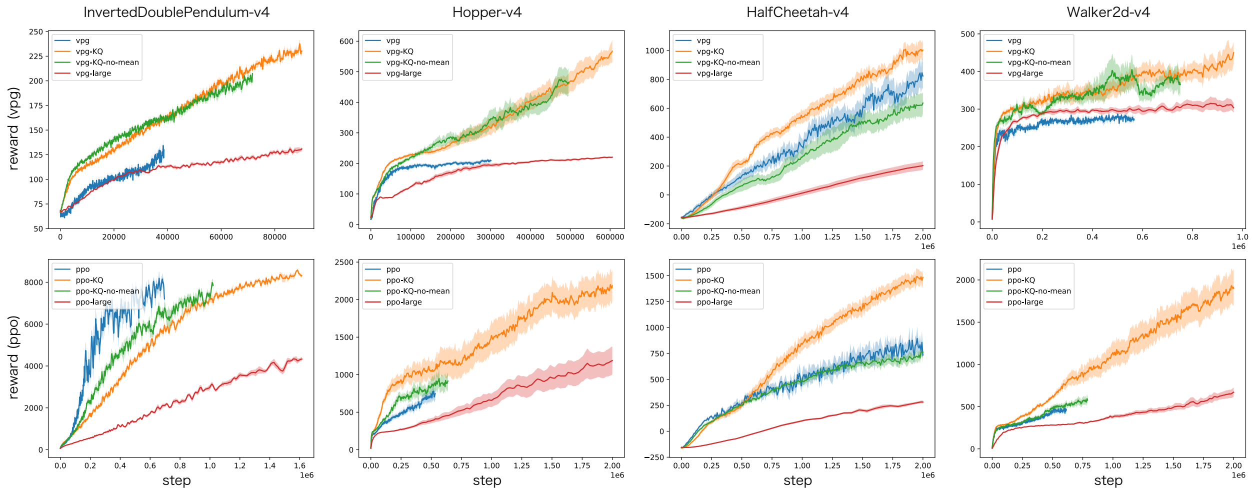

Figure 2: Learning curves in Mujoco Tasks: reward vs step