Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?

Andrew Ilyas*[1], Logan Engstrom*[1], Shibani Santurkar[1], Dimitris Tsipras[1],
Firdaus Janoos[2], Larry Rudolph[1,2], and Aleksander Mądry[1]

[1]MIT [2]Two Sigma
{ailyas,engstrom,shibani,tsipras,madry}@mit.edu
rudolph@csail.mit.edu, firdaus.janoos@twosigma.com

Abstract

We study how the behavior of deep policy gradient algorithms reflects the conceptual framework motivating their development. We propose a fine-grained analysis of state-of-the-art methods based on key aspects of this framework: gradient estimation, value prediction, optimization landscapes, and trust region enforcement. We find that from this perspective, the behavior of deep policy gradient algorithms often deviates from what their motivating framework would predict. Our analysis suggests first steps towards solidifying the foundations of these algorithms, and in particular indicates that we may need to move beyond the current benchmark-centric evaluation methodology.

1 Introduction

Deep reinforcement learning (RL) is at the core of some of the most publicized achievements of modern machine learning [26, 14, 2, 15]. To many, this framework embodies the promise of the real-world impact of machine learning. However, the deep RL toolkit has not yet attained the same level of engineering stability as, for example, the current deep (supervised) learning framework. Indeed, recent studies [4] demonstrate that state-of-the-art deep RL algorithms suffer from oversensitivity to hyperparameter choices, lack of consistency, and poor reproducibility.

This state of affairs suggests that it might be necessary to re-examine the conceptual underpinnings of deep RL methodology. More precisely, the overarching question that motivates this work is:

To what degree does the current practice of deep RL reflect the principles that informed its development?

The specific focus of this paper is on deep policy gradient methods, a widely used class of deep RL algorithms. Our goal is to explore the extent to which state-of-the-art implementations of these methods succeed at realizing the key primitives of the general policy gradient framework.

1.1 Our contributions

We begin by examining a prominent deep policy gradient method, proximal policy optimization (PPO) [25]. We find that PPO's performance depends heavily on optimizations not part of the core algorithm. These findings suggest that the practical success of PPO might not be possible to explain using its motivating theoretical framework.

This observation prompts us to take a broader look at policy gradient algorithms and their relation to their underlying framework. With this perspective in mind, we perform a fine-grained examination of key RL primitives as they manifest in practice.

*Equal contribution (ordered by random coin flip). Work done in part as an intern at Two Sigma.

Concretely, we study:

- **Gradient Estimation:** we find that even while agents are improving in terms of reward, the gradient estimates used to update their parameters are often poorly correlated with the true gradient.[1] We also find that gradient estimate quality decays with training progress and task complexity.
- **Value Prediction:** our experiments indicate that value networks successfully solve the supervised learning task they are trained on, but do not fit the true value function. Additionally, employing a value network as a baseline function only marginally decreases the variance of gradient estimates compared to using true value as a baseline (but dramatically increases agent's performance compared to using no baseline at all).
- **Optimization Landscapes:** we also observe that the optimization landscape induced by modern policy gradient algorithms is often not reflective of the underlying true reward landscape, and that the latter is often poorly behaved in the relevant sample regime.
- **Trust Regions:** our findings show that deep policy gradient algorithms sometimes violate theoretically motivated trust regions. In fact, in proximal policy optimization, these violations stem from a fundamental problem in the algorithm's design.

We believe that the above issues and our lack of understanding thereof majorly contribute to the widely observed brittleness and poor reproducibility of deep RL. This suggests that building reliable deep RL algorithms requires moving past benchmark-centric evaluations to a multi-faceted understanding of their often unintuitive behavior. Our work uncovers several areas where such understanding is most critically needed (c.f. Section 6).

2 Related Work

The idea of using gradient estimates to update neural network–based RL agents dates back at least to the work of Williams [30], who proposed the REINFORCE algorithm. Later, Sutton et al. [28] established a unifying framework that casts the previous algorithms as instances of the policy gradient method.

Our work focuses on proximal policy optimization (PPO) [25] and trust region policy optimization (TRPO) [22], which are two of the most prominent policy gradient algorithms used in deep RL. Much of the original inspiration for the usage of the trust regions stems from the conservative policy update of Kakade [7]. This policy update, similarly to TRPO, uses a natural gradient descent-based greedy policy update. TRPO also bears similarity to the relative policy entropy search method of Peters et al. [16], which constrains the distance between marginal action distributions (whereas TRPO constrains the conditionals of such action distributions).

A number of recent works – most notably, Henderson et al. [4] – documents the brittleness of the state-of-the-art deep RL algorithms. Rajeswaran et al. [17] and Mania et al. [13] also demonstrate that on many of the benchmark tasks, the performance of PPO and TRPO can be matched by fairly elementary randomized search approaches. Additionally, Tucker et al. [29] showed that one of the recently proposed extensions of the policy gradient framework, i.e., the usage of baseline functions that are also action-dependent (in addition to being state-dependent), might not lead to better policies after all.

3 Background

In the reinforcement learning (RL) setting, an agent interacts with a stateful environment with the goal of maximizing cumulative reward. Formally, we model the environment as a (possibly randomized) function mapping its current state s and an action a supplied by the agent to a new state s' and a resulting reward r. The choice of actions of the agent is governed by the its *policy* π. This policy is a function mapping

[1] It is important to note that this alone does not preclude such gradient signals from being useful. Indeed, this is known to be the case in many classical stochastic optimization settings. However, it is unclear to what degree we can transfer intuitions from these settings to the deep RL regime. Furthermore, we find that the effects of gradient variance may interact in complex ways with other factors.

environment states to a distribution over the actions to take. The objective of an RL algorithm is to find a policy π which maximizes the expected cumulative reward, where the expectation is taken over both environment randomness and the (randomized) action choices.

Preliminaries and notation. For a given policy π, we denote by $\pi(a|s)$ the probability that this policy assigns to taking action a when the environment is in the state s. We use $r(s,a)$ to denote the reward that the agent earns for playing action a in response to the state s. A *trajectory* $\tau = \{(a_t, s_t) : t \in \{1 \dots T\}\}$ is a sequence of state-action pairs that constitutes a valid transcript of interactions of the agent with the environment. (Here, a_t (resp. s_t) corresponds to the action taken by the agent (resp. state of the environment) in the t-th round of interaction.) We then define $\pi(\tau)$ to be the probability that the trajectory τ is executed if the agent follows policy π (provided the initial state of the environment is s_1). Similarly, $r(\tau) = \sum_t r(s_t, a_t)$ denotes the cumulative reward earned by the agent when following this trajectory, where s_t (resp. a_t) denote the t-th state (resp. action) in the trajectory τ. In the RL setting, however, we often choose to maximize the *discounted* cumulative reward of a policy $R := R_1$, where R_t is defined as

$$R_t(\tau) = \sum_{t'=t}^{\infty} \gamma^{(t'-t)} r_{t'} \ .$$

and $0 < \gamma < 1$ is a "discount factor". The discount factor ensures that the cumulative reward of a policy is well-defined even for an infinite time horizon, and it also incentivizes achieving reward earlier.

Policy gradient methods. A widely used class of RL algorithms that will be the focus of our analysis is the class of so-called *policy gradient methods*. The central idea behind these algorithms is to first parameterize the policy π_θ using a parameter vector θ. (In the deep RL context, π_θ is expressed by a neural network with weights θ.) Then, we perform stochastic gradient ascent on the cumulative reward with respect to θ. In other words, we want to apply the stochastic ascent approach to our problem:

$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] \ , \tag{1}$$

where $\tau \sim \pi_\theta$ represents trajectories (rollouts) sampled from the distribution induced by the policy π_θ. This approach relies on the key observation [28] that under mild conditions, the gradient of our objective can be written as:

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log(\pi_\theta(\tau))\ r(\tau)], \tag{2}$$

and the latter quantity can be estimated directly by sampling trajectories according to the policy π_θ.

When we use the discounted variant of the cumulative reward and note that the action of the policy at time t cannot affect its performance at earlier times, we can express our gradient estimate as:

$$\widehat{g_\theta} = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{(s_t,a_t)\in\tau} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot Q_{\pi_\theta}(s_t, a_t)\right] \ , \tag{3}$$

where $Q_{\pi_\theta}(s_t, a_t)$ represents the expected returns after taking action a_t from state s_t:

$$Q_{\pi_\theta}(s_t, a_t) = \mathbb{E}_{\pi_\theta}[R_t | a_t, s_t] \ . \tag{4}$$

Value estimation and advantage. Unfortunately, the variance of the expectation in (3) can be (and often is) very large, which makes getting an accurate estimate of this expectation quite challenging. To alleviate this issue, a number of variance reduction techniques have been developed. One of the most popular such techniques is the use of a so-called baseline function, wherein a state-dependent value is subtracted from Q_{π_θ}. Thus, instead of estimating (3) directly, we use:

$$\widehat{g_\theta} = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{(s_t,a_t)\in\tau} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot (Q_{\pi_\theta}(s_t, a_t) - b(s_t))\right], \tag{5}$$

where $b(\cdot)$ is a baseline function of our choice.

A natural choice of the baseline function is the value function, i.e.

$$V_{\pi_\theta}(s_t) = \mathbb{E}_{\pi_\theta}[R_t|s_t] \; . \tag{6}$$

When we use the value function as our baseline, the resulting gradient estimation problem becomes:

$$\widehat{g}_\theta = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{(s_t,a_t)\in\tau} \nabla_\theta \log \pi_\theta(a_t|s_t)\cdot A_{\pi_\theta}(s_t,a_t)\right], \tag{7}$$

where

$$A_{\pi_\theta}(s_t,a_t) = Q_{\pi_\theta}(s_t,a_t) - V_{\pi_\theta}(s_t) \tag{8}$$

is referred to as the *advantage* of performing action a_t. Different methods of estimating V_{π_θ} have been proposed, with techniques ranging from moving averages to the use of neural network predictors [23].

Surrogate Reward. So far, our focus has been on extracting a good estimate of the gradient with respect to the policy parameters θ. However, it turns out that directly optimizing the cumulative rewards can be challenging. Thus, a modification used by modern policy gradient algorithms is to optimize a "surrogate reward" instead. We will focus on maximizing the following local approximation of the true reward [22]:

$$\max_\theta \; \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}A_\pi(s_t,a_t)\right] \qquad \Big(= \mathbb{E}_{\pi_\theta}\left[A_\pi\right]\Big), \tag{9}$$

or the normalized advantage variant proposed to reduce variance [25]:

$$\max_\theta \; \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}\widehat{A}_\pi(s_t,a_t)\right] \tag{10}$$

where

$$\widehat{A}_\pi = \frac{A_\pi - \mu(A_\pi)}{\sigma(A_\pi)} \tag{11}$$

and π is the current policy.

Trust region methods. The surrogate reward function, although easier to optimize, comes at a cost: the gradient of the surrogate reward is only predictive of the policy gradient locally (at the current policy). Thus, to ensure that our update steps we derive based on the surrogate reward are predictive, they need to be confined to a "trust region" around the current policy. The resulting trust region methods [7, 21, 25] try to constrain the local variation of the parameters in policy-space by restricting the distributional distance between successive policies.

A popular method in this class is trust region policy optimization (TRPO) [22], which constrains the KL divergence between successive policies on the optimization trajectory, leading to the following problem:

$$\begin{aligned} \max_\theta \quad & \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}\widehat{A}_\pi(s_t,a_t)\right] \\ \text{s.t.} \quad & D_{KL}(\pi_\theta(\cdot \mid s)||\pi(\cdot \mid s)) \le \delta, \quad \forall s \; . \end{aligned} \tag{12}$$

In practice, this objective is maximized using a second-order approximation of the KL divergence and natural gradient descent, while replacing the worst-case KL constraints over all possible states with an approximation of the mean KL based on the states observed in the current trajectory.

Proximal policy optimization. In practice, the TRPO algorithm can be computationally costly—the step direction is estimated with nonlinear conjugate gradients, which requires the computation of multiple Hessian-vector products. To address this issue, Schulman et al. [25] propose proximal policy optimization (PPO), which utilizes a different objective and does not compute a projection. Concretely, PPO proposes replacing the KL-constrained objective (12) of TRPO by clipping the objective function directly as:

$$\max_{\theta} \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\min\left(\text{clip}\left(\rho_t, 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_\pi(s_t,a_t),\ \rho_t\widehat{A}_\pi(s_t,a_t)\right)\right] \tag{13}$$

where

$$\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)} \tag{14}$$

In addition to being simpler, PPO is intended to be faster and more sample-efficient than TRPO [25].

4 Implementation Matters in Policy Gradient Methods

Our overarching goal is to understand how the conceptual principles of policy gradient methods are reflected in practice. A natural point of start, therefore, is an investigation of the practice (that is, the implementation) of state-of-the-art policy gradient methods.

It turns out that even the canonical implementations of these methods contain additional, non-trivial optimizations that do not constitute core parts of the respective algorithms. A prominent example here is the PPO algorithm [25] described above. Specifically, the standard implementation of PPO[2] contains the following additional optimizations (among others, laid out in Appendix 9.3):

1. **Value function clipping:** Although Schulman et al. [25] originally suggest fitting the value network via regression to target values:
$$L^V = (V_{\theta_t} - V_{targ})^2,$$
in the standard implementation the value network is instead fit with a PPO-like objective:
$$L^V = \min\left[(V_{\theta_t} - V_{targ})^2, \left(\text{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{targ}\right)^2\right],$$
where V_θ is clipped around the previous value estimates (and ε is fixed to the same value as the value used in (13) to clip the probability ratios).

[2]From the OpenAI baselines GitHub repository: `https://github.com/openai/baselines`

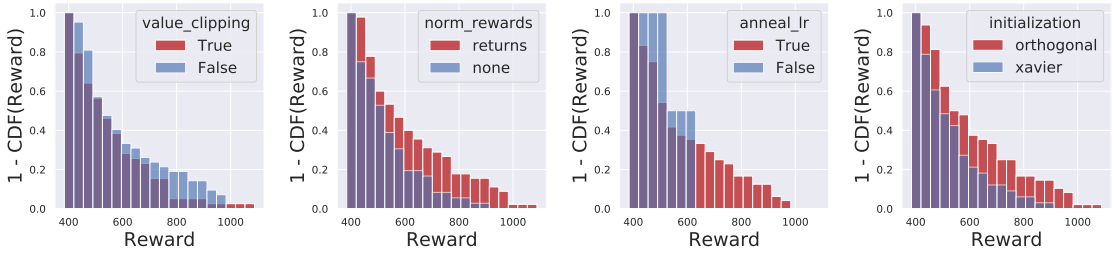


Figure 1: An ablation study on the four optimizations described in Section 4 (value clipping, reward scaling, network initialization, and learning rate annealing). For each of the 2^4 possible configurations of optimizations, we train a Humanoid-v2 agent using PPO with three random seeds and multiple learning rates, and choose the learning rate which gives the best average reward (over the three random seeds). We then consider all rewards from the "best learning rate" runs (a total of 3×2^4 agents), and plot histograms in which agents are partitioned based on whether each optimization is on or off. Our results show that reward normalization, Adam annealing, and network initialization are crucial to obtaining the best average reward with PPO.

2. **Reward scaling:** Rather than feeding the rewards directly from the environment into the objective, the PPO implementation performs a certain discount-based scaling scheme. In this scheme, the rewards are divided through by the standard deviation of a rolling discounted sum of the rewards (without subtracting and re-adding the mean)—see Algorithm 1 in Appendix 9.3.

3. **Orthogonal initialization and layer scaling:** Instead of using the default weight initialization scheme for the policy and value networks, the implementation uses an orthogonal initialization scheme with scaling that varies from layer to layer.

4. **Adam learning rate annealing:** Depending on the task, the implementation sometimes anneals the learning rate of Adam [11] for optimization.

These optimizations turn out to have a dramatic effect on the performance of PPO. To demonstrate this, we perform a full ablation study on the four optimizations mentioned above[3]. Figure 1 shows a histogram of the final rewards of agents trained with every possible configuration of the above optimizations. Our findings suggest that these optimizations are necessary for PPO to attain its claimed performance.

Our ability to understand PPO from an algorithmic perspective hinges on being able to distill its fundamental principles from such implementation details. To this end, we consider a variant of PPO called PPO-Minimal (PPO-M) which implements only the core of the algorithm. PPO-M uses the standard value network loss, no reward scaling, the default network initialization, and Adam with a fixed learning rate. We then explore PPO-M alongside PPO and TRPO.

5 Examining the Primitives of Deep Policy Gradient Algorithms

As we observed in the previous section, the performance of state-of-the-art deep policy gradient algorithms may crucially rely on factors outside of the core RL framework. In this section, we aim to investigate the degree to which our theoretical understanding of RL applies to modern methods. To this end, we consider key primitives of policy gradient algorithms: gradient estimation, value prediction, reward fitting, and trust region enforcement. In what follows, we perform a fine-grained analysis of state-of-the-art policy gradient algorithms through the lens of these primitives.

5.1 Gradient estimate quality

A central premise of policy gradient methods is that stochastic gradient ascent on a suitable objective function yields a good policy. In particular, recall from Section 3 that these algorithms use as a primitive the gradient of the (surrogate) reward function:

$$\hat{g} = \nabla_\theta \mathbb{E}_{(s_t,a_t)\sim\pi_0}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_0(a_t|s_t)}\widehat{A}_{\pi_0}(s_t,a_t)\right] = \mathbb{E}_{(s_t,a_t)\sim\pi_0}\left[\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi_0(a_t|s_t)}\widehat{A}_{\pi_0}(s_t,a_t)\right] \tag{15}$$

An underlying assumption behind the theory of these methods is that we have access to a reasonable estimate of this quantity. This assumption effectively translates into an assumption that we can accurately estimate the expectation above using an empirical mean of finite (typically $\sim 10^3$) samples. Evidently (since the agent attains a high reward), the signal from these estimates is sufficient to consistently improve reward—we are thus interested in the quality of these gradient estimates in practice.

How accurate are the gradient estimates we compute? To answer this question, we examine two of the most natural measures of estimate quality: the empirical variance and the convergence to the "true" gradient. To evaluate the former, we measure the average pairwise cosine similarity between estimates of the gradient computed from the same policy with independent rollouts (Figure 2). We evaluate the latter by first forming an estimate of the true gradient with a large number of state-action pairs. We then examine the convergence of gradient estimates to this "true" gradient (which we once again measure using cosine similarity) as we increase the number of samples (Figure 3).

[3]Due to restrictions on computational resources, it was not possible to perform a full study on all of the optimization including those from Appendix 9.3

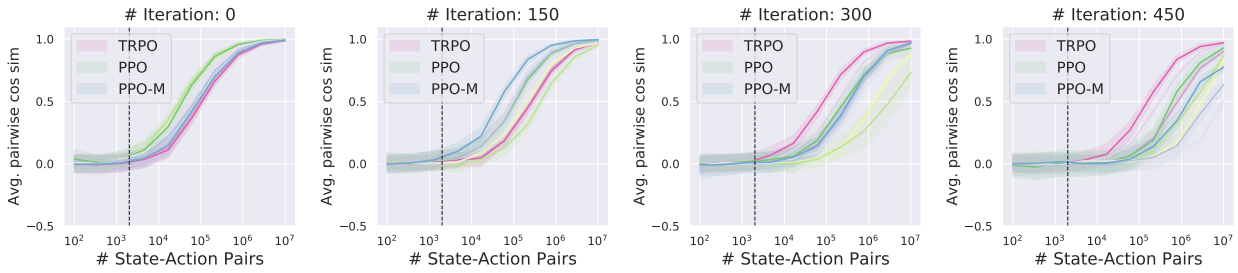


Figure 2: Empirical variance of the estimated gradient (c.f. (15)) as a function of the number of state-action pairs used in estimation in the MuJoCo Humanoid task. We measure the average pairwise cosine similarity between ten repeated gradient measurements taken from the same policy, with the 95% confidence intervals (shaded). For each algorithm, we perform multiple trials with the same hyperparameter configurations but different random seeds, shown as repeated lines in the figure. The vertical line (at $x = 2\text{K}$) indicates the sample regime used for gradient estimation in standard implementations of policy gradient methods. In general, it seems that obtaining tightly concentrated gradient estimates would require significantly more samples than are used in practice, particularly after the first few timesteps. For other tasks – such as Walker2d-v2 and Hopper-v2 – the plots (seen in Appendix Figure 11) have similar trends, except that gradient variance is slightly lower.

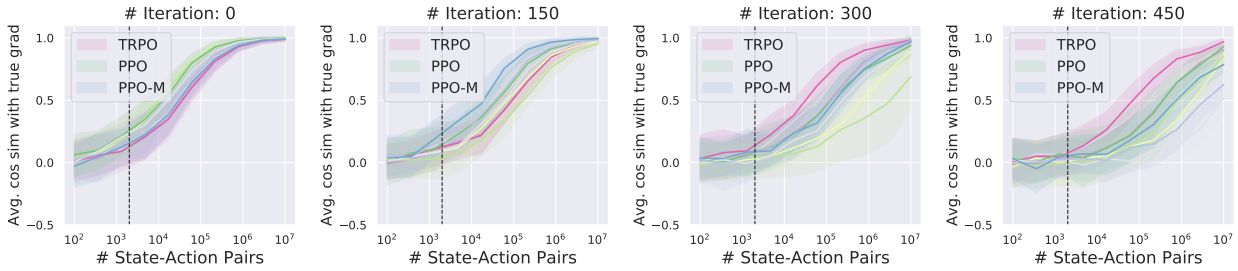


Figure 3: Convergence of gradient estimates (c.f. (15)) to the "true" expected gradient in the MuJoCo Humanoid task. We measure the mean cosine similarity between the "true" gradient approximated using ten million state-action pairs, and ten gradient estimates which use increasing numbers of state-action pairs (with 95% confidence intervals). For each algorithm, we perform multiple trials with the same hyperparameter configurations but different random seeds. The vertical line (at $x = 2\text{K}$) indicates the sample regime used for gradient estimation in standard implementations of policy gradient methods. Observe that although it is possible to empirically estimate the true gradient, this requires several-fold more samples than are used commonly in practical applications of these algorithms. See additionally that the estimation task becomes more difficult further into training. For other tasks – such as Walker2d-v2 and Hopper-v2 – the plots (seen in Appendix Figure 12) have similar trends, except that gradient estimation is slightly better.

We observe that *deep policy gradient methods operate with poor estimates of the gradient.* Although it is possible to obtain accurate estimates of the gradient, doing so requires significantly more samples than what current implementations use. This effect becomes stronger as task complexity increases and as training progresses. (Contrast Humanoid-v2 ("hard" task) to other tasks and successive checkpoints in Figures 2 and 3.) In fact, we sometimes observe a zero or even negative correlation in the relevant sample regime.

What is the variance of policy steps? Gradient estimates are the key primitive used in computing steps in policy gradient methods. Given that gradient estimates tend to have low accuracy and high variance in the relevant sampling regime, it is natural to examine whether the actual steps exhibit similar behavior. Indeed, our results (Figure 13 in Appendix 9.4) indicate that the policy steps do have high empirical variance: on the Walker2d and Humanoid MuJoCo tasks, for example, the mean pairwise cosine similarity between steps sometimes even hovers around zero (at later iterations).

Discussion. We have observed that RL agents function in a sample regime where gradient estimates are often poorly correlated. However, the consistent improvements in agents' reward are an evidence that these estimates do convey nontrivial signal. Indeed, these results may be reminiscent of similar phenomena in stochastic optimization, where often seemingly small correlations with the true gradient are enough to successfully optimize an objective. In convex optimization, this problem of "learning with noisy gradients" has rigorous theoretical explanation [19, 1]. And, in supervised deep learning, even if the lack of smoothness and convexity makes attaining a full theoretical understanding difficult, there exists a substantial body of work dedicated to empirically studying how stochastic gradients impact optimization and the structure of the loss landscape in general [9, 20, 12, 10, 6]. In contrast, in the deep RL setting:

- We lack a crisp understanding of the structure of the deep RL reward landscape and its critical points. (We revisit this issue in Section 5.3.)
- While the total number of samples (state-action pairs) may seem quite large, the number of *independent* samples used at each iteration corresponds to the number of complete trajectories. This quantity (a) tends to be much lower than the number of state-action pairs, and (b) varies across iterations during training. This issue is further complicated by the fact that the standard optimization objective (the surrogate loss) uses normalized statistics (the advantage) which are computed across all the trajectories.
- Overall, the sample complexity of the gradient estimates (which can be seen as a parallel to batch size in supervised learning) seems to have a profound impact on the stability of training agents. In particular, many of the issues reported by Henderson et al. [4] are claimed to disappear [27] in higher sample-complexity regimes. This might imply that the algorithms we analyze operate at a cusp of a "noise barrier", i.e., we work in a regime where we use close to the minimum number of samples required for convergence. Understanding the implications of working in this sample regime, and more generally the impact of sample complexity on training stability remains to be precisely understood.
- Due to the presence of the value network (which we discuss more in the following section), training an agent with deep policy gradient algorithms entails training two interacting neural networks. This makes the corresponding optimization landscape and training dynamics even more difficult to understand.

All the above factors make it unclear to what degree our intuition from classical settings can transfer to the deep RL regime. And the policy gradient framework, as of now, provides little insight into the variance of gradient estimates and its impact on reward optimization.

5.2 Value prediction

Our study in the previous section found that the gradient estimates we compute and use in modern policy gradient methods tend to be rather noisy. This motivates a deeper look into the gradient estimation pipeline. After all, the policy gradient in its original formulation (c.f. (2)) is known to be hard to estimate, and thus algorithms employ a variety of variance reduction techniques. The most popular of these techniques is the use of a baseline function. Recall from Section 3 that an equivalent form of the policy gradient is given by:

$$\widehat{g}_\theta = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{(s_t,a_t)\in\tau} \nabla_\theta \log \pi_\theta(a_t|s_t)\cdot(Q_{\pi_\theta}(s_t,a_t) - b(s_t))\right] \tag{16}$$

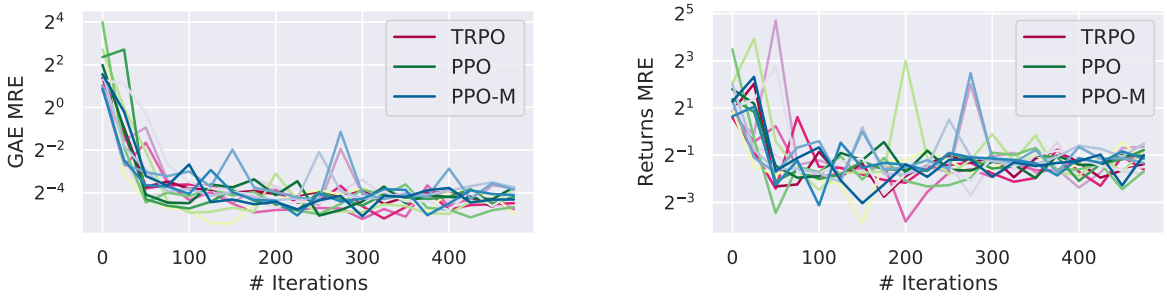
where $b(s_t)$ is some fixed function of the state s_t. A canonical choice of baseline function is the value function $V_\pi(s)$ (c.f. (6)). Indeed, fitting a value-estimating function (a neural network, in the deep RL setting) and using it as a baseline function is precisely the approach taken by most deep policy gradient methods. Concretely, one trains a value network $V^\pi_{\theta_t}$ such that:

$$\theta_t = \min_\theta \ \mathbb{E}\left[\left(V^\pi_\theta(s_t) - (V^\pi_{\theta_{t-1}} + A_t)\right)^2\right] \tag{17}$$

where $V^\pi_{\theta_{t-1}}$ are the estimates given by the last value function, and A_t is the advantage of the policy (c.f. (8)) with respect to these estimated values. (Typically, A_t is estimated using generalized advantage estimation, as described in [24]). Our findings in the previous section thus prompt us to take a closer look at the value network and its impact on the variance of gradient estimates.

Value prediction as a supervised learning problem. We first analyze the value network through the lens of the supervised learning problem it solves. After all, (17) describes an empirical risk minimization problem, where a loss function is minimized over a set of sampled training datapoints (s_t, a_t). So, how does V_θ^π perform as a solution to (17)? And in turn, how does (17) perform as a proxy for learning the true value function?

Our results (Figure 4a) show that the value network *does* succeed at both fitting the given loss function and generalizing to unseen data, showing low and stable mean relative error (MRE). However, the significant drop in performance as shown in Figure 4b indicates that the supervised learning problem induced by (17) *does not* lead to V_θ^π learning the underlying true value function.



(a) (b)

Figure 4: Quality of value prediction in terms of mean relative error (MRE) on heldout state-action pairs for agents trained to solve the MuJoCo Walker2d-v2 task. We observe in (a) that the agents do indeed succeed at solving the supervised learning task they are trained for – the validation MRE on the GAE-based value loss $(V_{old} + A_{GAE})^2$ (c.f. (17)) is small. On the other hand, in (b) we see that the returns MRE is still quite high – the learned value function is off by about 50% with respect to the underlying true value function. Similar plots for the training set, and other MuJoCo tasks are in Appendix Figures 14 and 15.

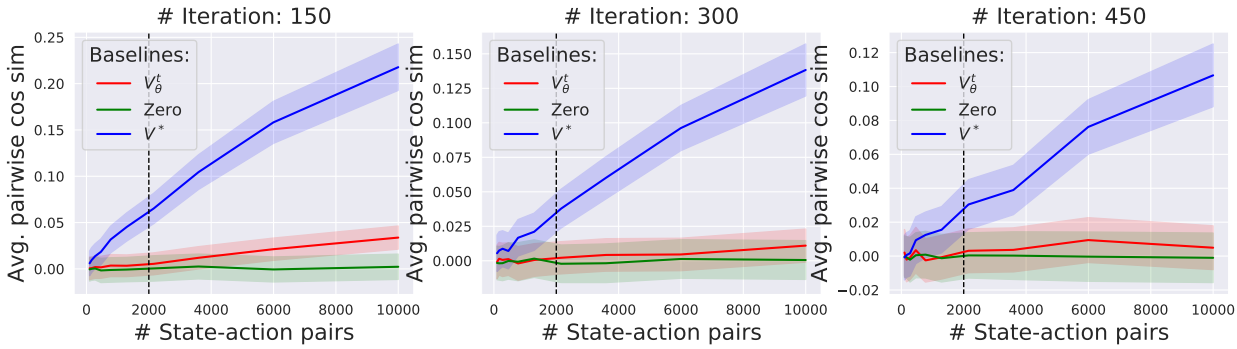


Figure 5: Efficacy of the value network as a variance reducing baseline function for agents trained on the Walker2d-v2 MuJoCo task. We measure the empirical variance of the gradient (c.f. (15)) as a function of the number of state-action pairs used in estimation, for different choices of baseline functions. Specifically, we compare the baseline coming from: the value network (used by the agent during training), the "true" value function (fit directly to the returns using a large number of state-action pairs (500K) sampled from the *current* policy) and the "zero" value function (i.e. simply replacing advantages with returns). We observe that the usage of the true value function leads to a significantly lower-variance estimate of the gradient as compared to the value network. In turn, employing the value network yields a noticeably larger variance reduction compared to the zero baseline function, even though this difference may appear rather small in the most relevant, small-sample regime (2K). Additional plots can be found in Appendix Figure 16.

Does the value network lead to a reduction in variance? Though evaluating the V_θ^π baseline function as a value predictor as we did above is informative, in the end the sole purpose of the value function is to reduce variance. Thus, our question becomes: how does using our value function actually impact the variance of our gradient estimates? To this end, we compare the variance reduction that results from employing our value network against both an "true" value function, and a trivial "zero" baseline function (i.e. simply replacing advantages with returns). Our results, captured in Figure 5, show that the "true" value function yields a much lower-variance estimate of the gradient. This is especially true in the sample regime in which we operate. We note, however, that despite not effectively predicting the true value function or inducing the same degree of variance reduction, the value network *does* help (compared to the "zero" baseline). Additionally, the seemingly marginal increase in gradient correlation provided by the value network (compared to the "true" baseline function) turns out to result in a significant improvement in agent performance. (Indeed, our experiments show that agents trained without a baseline function attain almost an order of magnitude worse final reward.)

Our findings here suggest that we still need a better understanding of the role of the value network in agent training and raise several questions that we discuss in Section 6.

5.3 Exploring the optimization landscape

Another fundamental assumption of policy gradient algorithms is that applying first-order updates with respect to the policy parameters yields better-performing policies. It is thus natural to examine how valid this assumption is.

The landscape of true rewards. We begin by examining the landscape of agent reward with respect to the policy parameters. Indeed, even if deep policy gradient methods do not optimize for the true reward directly (as they work, e.g., with the surrogate reward – c.f. (9)), the ultimate goal of any policy gradient algorithm is to navigate this landscape. Figure 6 indicates that estimating the true reward landscape with a high number of samples yields a relatively smooth reward landscape, perhaps suggesting viability of direct optimization on the reward landscape. However, Figure 6 also shows that estimating the true reward landscape in the typical, low sample regime results in a landscape that appears jagged and poorly-behaved. The low-sample regime thus gives rise to a certain kind of barrier to direct reward optimization. Indeed, applying our algorithms in this regime makes it impossible to distinguish between good and bad points in the landscape, even though the true underlying landscape is fairly well-behaved.

The landscape of surrogate rewards. The observed untamed nature of the rewards landscape has led to the development of alternate approaches to reward maximization. Recall that an important element of many modern policy gradient methods is the maximization of a surrogate reward function in place of the true rewards. The surrogate reward, based on a relaxation of the policy improvement theorem of Kakade and Langford [8], can be viewed as a simplification of the reward maximization objective (c.f. Section 3).

As a purported approximation of the true returns, one would expect that the surrogate reward landscape approximates the true reward landscape fairly well. That is, parameters corresponding to good surrogate reward will also correspond to good true reward.

Figure 7 shows that the early stages of training the optimization landscapes of the true and surrogate reward are indeed approximately aligned. However, as training progresses, the surrogate reward becomes much less predictive of the true reward in the low-sample regime modern algorithms are typically applied in. In particular, we often observe that directions that *increase* the surrogate reward lead to a *decrease* of the true reward (see Figures 7, 8). This suggests that the objective optimized at each step by the algorithm is only weakly related to the objective we actually care about. In a higher-sample regime (using several orders of magnitude more samples), we find that PPO and TRPO turn out to behave rather differently. In the case of TRPO, the update direction leads to a surrogate reward that matches the true reward much more closely. However, in the case of PPO (and PPO-M) we consistently observe landscapes where the step direction leads to lower true reward, even in the high-sample regime. This suggests that even when estimated accurately enough, the surrogate reward might not be a suitable proxy for the true reward. (Recall that, as seen in Section 5.1, this is a sample regime where we *are* able to estimate the true gradient of the reward fairly well.)

5.4 Optimization over the trust region

The exploration of the landscape we performed in the previous section showed that in the low-sample regime (which is the regime that most state-of-the-art methods work in), the true reward changes fairly sharply when we move in the direction of the update step. This implies that taking even small steps in parameter space can lead to a significant drop in reward. A natural way to deal with this noisy, highly varied landscape is to thus enforce a "trust region" of some kind. This ensures that a single step does not make the policy stray too far from the previous one. Indeed, there is a particular way to enforce such a trust region that comes with theoretical guarantees. Specifically, Schulman et al. [22] show that constraining maximum KL divergence (i.e. $\max_s D_{KL}(\pi_\theta(\cdot|s)||\pi_0(\cdot|s))$) leads to a monotonic per-step improvement guarantee for expected return. In practice, however, one tends to enforce a looser trust region constraint. This is often motivated by the desire to circumvent both computational constraints, and the inherent pessimism of worst-case bounds. We thus want to examine the potential discrepancy between the theoretically motivated trust region, and the ones we actually impose.

In what sense do deep policy gradient algorithms enforce a trust region? We begin by checking if the (relaxed) trust region constraints maintained by our policy gradient algorithms happen to also enforce the theoretically motivated trust region based on maximum KL. In the case of TRPO, the trust region we maintain is based on the mean KL divergence. While this technically does translate to a bound on the maximum KL divergence (via non-negativity of KL), in practice we find that the maximum KL divergence is several orders of magnitude larger than our enforced bound on mean KL, as shown in Figure 9. (Interestingly, the mean-KL constraint does generalize to unseen set of trajectories (Appendix 29), suggesting that the trust region constraint does not overfit to the trajectories over which it is computed.) On the other hand, PPO-M *does not* seem to enforce a mean-KL trust region (and by extension, a max-KL trust region). This is surprising for at least two reasons. First, it turns out that PPO *does* enforce such a trust region, which implies that it must be one of the aforementioned additional optimizations (see Section 4), and not the core clipping technique of PPO that maintains this trust region. Secondly, the PPO(-M) clipping technique aims to directly bound the ratio $\pi_\theta(a|s)/\pi_0(a|s)$ for all state-action pairs (s, a), which should be sufficient to bound the maximum KL. In fact, we find that despite bounding the maximum of these ratios appearing to be a simpler goal, neither PPO nor TRPO effectively accomplish this. For TRPO, this actually makes

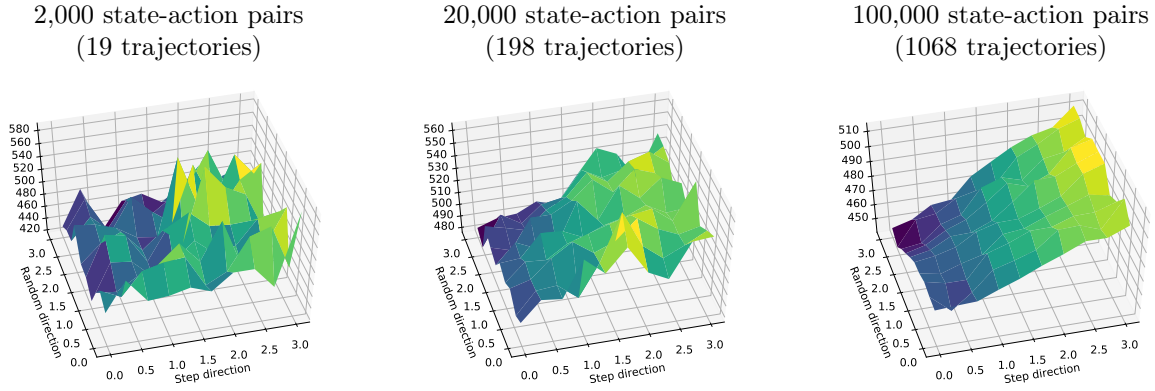


Figure 6: True reward landscape concentration for TRPO on the Humanoid-v2 MuJoCo task. We visualize the landscape at a particular training iteration (300) with different number of trajectories used to estimate the reward (each subplot), both in the direction of the step taken and a random direction. Moving one unit along the "step direction" axis corresponds to moving one full algorithm step in the parameter space. In the random direction, one unit corresponds to moving along a random Gaussian vector, rescaled to norm 2, in the parameter space. Note that in practice, the norm of the step is typically an order of magnitude lower than the random direction. We observe that the while landscape appears very noisy in the relevant small sample regime, using a large number of samples does reveal a smooth and well-behaved underlying landscape. See Figures 27, 26 of the Appendix for additional concentration plots.

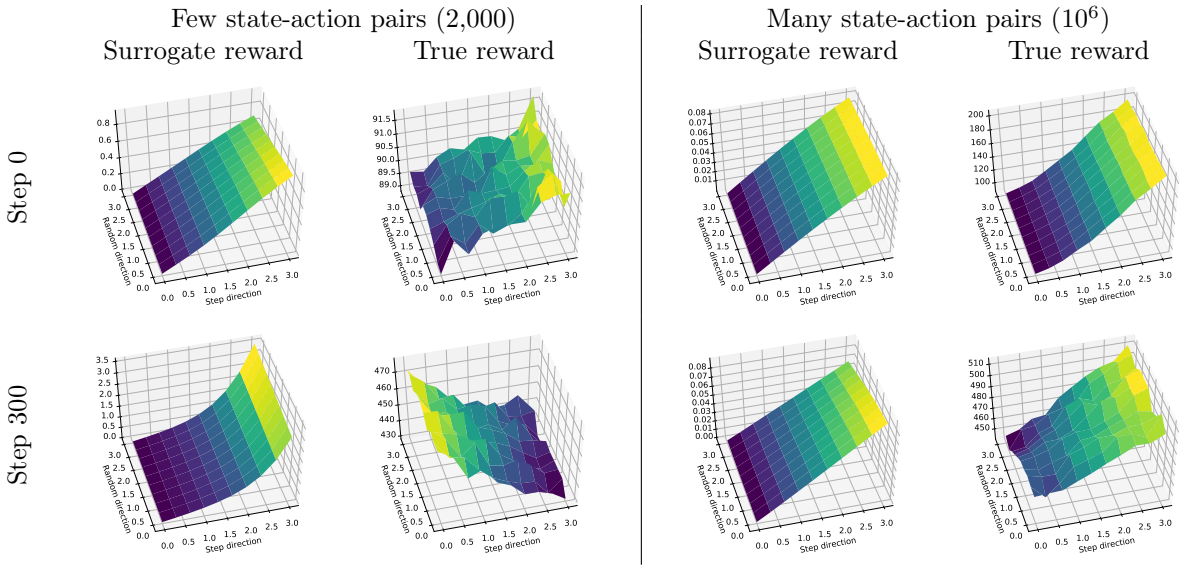


Figure 7: True and surrogate reward landscapes for TRPO on the Humanoid-v2 MuJoCo task. We visualize the landscape of the true and surrogate reward in the direction of the step taken and a random direction (similar to Figure 6). The surrogate reward corresponds to the actual function optimized by the algorithm at each step. The true reward landscape is estimated with 10^6 state-action pairs per point. We compare the landscapes at different points in training and with a different number or state-action pairs used to compute the update step. We observe that early in training the true and surrogate landscapes align fairly well. Later in training the TRPO landscapes in the high sample regime align well. The corresponding landscape for PPO can be found in Figure 8. Additional landscapes can be found in Figures 17-25 of the Appendix.

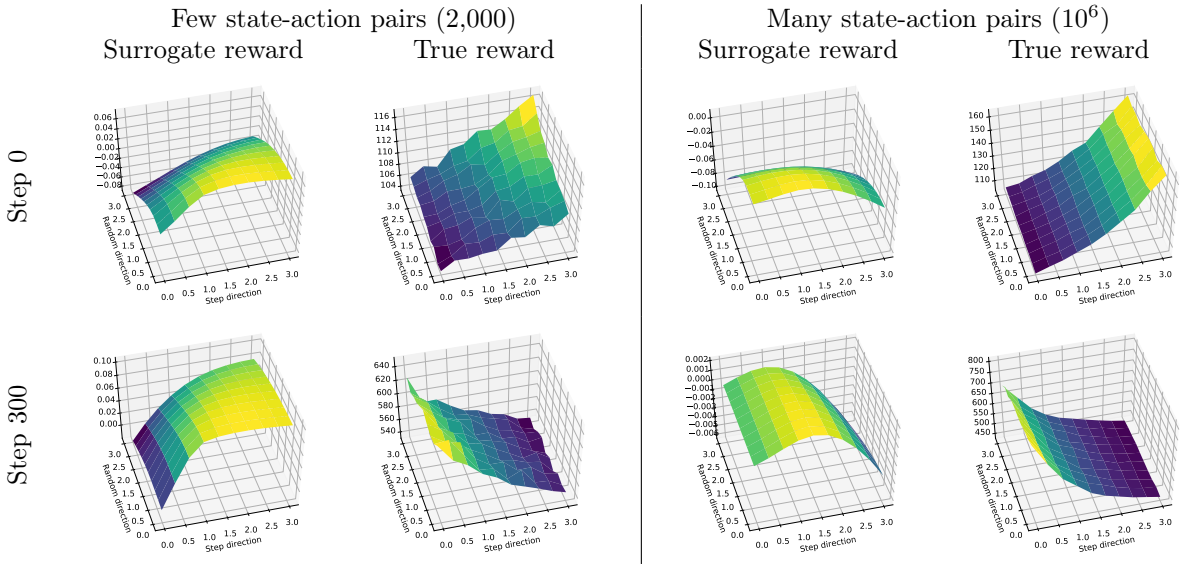


Figure 8: True and surrogate reward landscapes for PPO on the Humanoid-v2 MuJoCo task. See Figure 7 for a description. We observe that early in training the true and surrogate landscapes align fairly well. However, later in training increasing the surrogate reward leads to points with lower true reward.

sense: given any bound δ, there exist distributions p and q such that $D_{KL}(p, q) \leq \delta$ but the maximum ratio between them is unbounded.

In the case of PPO, however, the unbounded nature of the likelihood ratios is puzzling. In particular, the

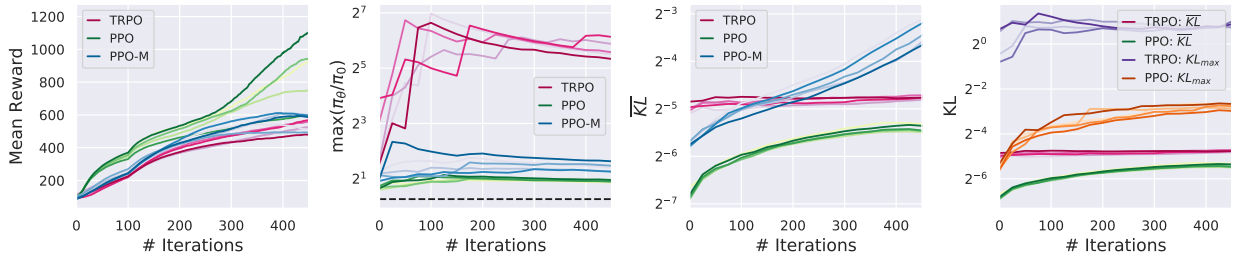


Figure 9: Per step mean reward, maximum ratio (c.f. (13)), mean KL, and maximum versus mean KL for agents trained to solve the MuJoCo Humanoid task. The quantities are measured over the state-action pairs collected in the *training step*. Each line represents a training curve from a separate agent. The black dotted line represents the $1+\epsilon$ ratio constraint in the PPO algorithm, and we measure each quantity every twenty five steps. From the plots we can see that the PPO variants' maximum ratios consistently violates the ratio "trust region." We additionally see that while PPO constrains the mean KL well, PPO-M does not, suggesting that the core PPO algorithm is not sufficient to maintain a ratio "trust region." We additionally measure the quantities over a *heldout* set of state-action pairs and find little qualitative difference in the results (seen in Figure 29 in the appendix), suggesting that TRPO does indeed enforce a mean KL trust region. We show plots for additional tasks in the Appendix in Figure 28.

main premise which motivates calling PPO a trust region method is the assumption that these likelihood ratios of the conditional distributions are controlled. The results depicted in Figure 9 thus prompt us to consider the PPO objective more carefully.

Clipping vs. constrained optimization. It turns out that despite an intuitively sound motivation (clipping in the PPO objective (13) aims to disincentive the policy from increasing a ratio past the given bound), *the PPO objective is fundamentally unable to enforce a trust region.* Indeed, the optimization problem PPO solves might have many optima, most of which are actually far outside the trust region:

Theorem 5.1 (Optima of the clipped objective; proof in Appendix 9.8)**.** *The optimization problem considered by proximal policy optimization (PPO), i.e.*

$$\min_{\{\pi_\theta(a|s)\}} \mathbb{E}_{(s,a)\in\tau\sim\pi}\left[clip\left(\frac{\pi_\theta(a|s)}{\pi(a|s)}, 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_\pi(s,a)\right],$$

either has (a) a unique solution that is identical to its unclipped counterpart, or (b) uncountably many optima, of which only one is within the trust region $[(1-\varepsilon)\pi, (1+\varepsilon)\pi]$.

Still, one could hope that the iterative first-order optimization scheme that PPO actually employs to solve its objective avoids these unfavorable optima, and thus successfully enforces the trust region. We note, however, that the gradient of the PPO objective is:

$$\nabla_\theta L_{PPO} = \begin{cases} \nabla_\theta L_\theta & \frac{\pi_\theta(a|s)}{\pi(a|s)} \in [1-\epsilon, 1+\epsilon] \text{ or } L_\theta^C < L_\theta \\ 0 & \text{otherwise} \end{cases},$$

where

$$L_\theta := \mathbb{E}_{(s,a)\in\tau\sim\pi}\left[\frac{\pi_\theta(a|s)}{\pi(a|s)}A_\pi(s,a)\right] \qquad \text{and} \qquad L_\theta^C := \mathbb{E}_{(s,a)\in\tau\sim\pi}\left[\text{clip}\left(\frac{\pi_\theta(a|s)}{\pi(a|s)}, 1-\varepsilon, 1+\varepsilon\right)A_\pi(s,a)\right]$$

are the standard and clipped versions of the surrogate reward, respectively. As a result, since we initialize π_θ as π (and thus the ratios start all equal to one), the first step we take is identical to a maximization step over the *unclipped* surrogate reward. Therefore, the size of step we take is determined solely be the steepness of the surrogate landscape (i.e. Lipschitz constant of the optimization problem we solve), and we can end up moving arbitrarily far from the trust region. (Recall that Figure 9 documents exactly this kind of behavior of PPO-M.)

6 Towards Stronger Foundations for Deep Reinforcement Learning

Deep reinforcement learning (RL) algorithms are rooted in a well-grounded framework of classical RL, and have shown great promise in practice. However, as our investigations uncover, this framework fails to explain much of the behavior of these algorithms. This disconnect impedes our understanding of why these algorithms succeed (or fail). It also poses a major barrier to addressing key challenges facing deep RL, such as widespread brittleness and poor reproducibility (cf. Section 4 and [4, 5]).

To close this gap, we need to either develop methods that adhere more closely to theory, or build theory that can capture what makes existing policy gradient methods successful. In both cases, the first step is to precisely pinpoint where theory and practice diverge. To this end, we analyze and consolidate our findings from the previous section.

Gradient estimation. Our analysis in Section 5.1 shows that the quality of gradient estimates that policy gradient algorithms use is rather poor. Indeed, even when agents are still improving, such gradient estimates are often poorly correlated (and sometimes even uncorrelated) with the true gradient (c.f. Figure 3), and with each other (c.f. Figure 2). We also note that gradient correlation decreases as training progresses and task complexity increases. While this certainly does not preclude the estimates from conveying useful signal (after all, this tends to be the case in classical stochastic optimization settings), the exact underpinnings of this phenomenon in the deep RL setting still elude us. In particular, in Section 5.1 we outline a few keys ways in which the deep RL setting is quite unique and difficult to understand from an optimization perspective. Understanding the relation between the quality of gradient estimate and optimization in deep RL from both the theoretical and practical standpoints is thus a challenging (and pressing) question.

Value prediction. The findings presented in Section 5.2 identify two key issues. First, while the value network successfully solves the supervised learning task it is trained on, it does not accurately model the "true" value function. Second, employing the value network as a baseline does decrease the gradient variance (compared to the trivial ("zero") baseline). However, this decrease is rather marginal compared to the variance reduction offered by the "true" value function.

It is natural to wonder whether this failure in modeling the true value function is inevitable. For example, how does the loss function used to train the value network actually impact value prediction and variance reduction? More broadly, we lack a crisp understanding of the precise role of the value network in training. Can we empirically quantify the relationship between variance reduction and performance? And does the value network play a broader role in policy gradient methods than just reducing the variance?

Optimization landscape. We have also seen, in Section 5.3, that the optimization landscape induced by modern policy gradient algorithms is often not reflective of the underlying true reward landscape. In fact, in the sample-regime where policy gradient methods operate, the true reward landscape is noisy and the surrogate reward is often misleading. We thus need a better understanding of why the current methods succeed despite these issues, and, more broadly, how to navigate the true reward landscape more accurately.

Trust region approximation. In general, our findings indicate that there may be a number of reasons why policies need to be locally similar. These include noisy gradient estimates, poor baseline functions and misalignment of the surrogate landscape. Not only is our theory surrounding trust region optimization oblivious to these factors, it is also notoriously difficult to translate this theory into efficient algorithms. Deep policy gradient methods thus resort to relaxations of trust region constraints, which makes their performance difficult to properly understand and analyze (c.f. Section 5.4). Therefore, we need either techniques that enforce trust regions more strictly, or a rigorous theory of trust region relaxations.

7 Conclusion

In this work, we analyze the degree to which key primitives of deep policy gradient algorithms follow their conceptual underpinnings. Our experiments show that these primitives often do not conform to the expected

behavior: gradient estimates poorly correlate with the true gradient, value networks reduce gradient estimation variance to a significantly smaller extent than the true value, the underlying optimization landscape can be misleading, and the update steps frequently violate trust regions. Though we focus on two of the most prominent deep policy gradient algorithms, the analysis we propose is generally applicable.

This demonstrates that there is a significant gap between the theory inspiring current algorithms and the actual mechanisms driving their performance. Overall, our findings suggest that developing a deep RL toolkit that is truly robust and reliable will require moving beyond the current benchmark-driven evaluation model to a more fine-grained understanding of deep RL algorithms.

8 Acknowledgments

We thank Ludwig Schmidt for valuable feedback on an earlier version of this work. We also thank the authors of [18] and [3] for inspiring the title of the paper and Section 4, respectively.

SS was supported by the National Science Foundation (NSF) under grants IIS-1447786, IIS-1607189, and CCF-1563880, and the Intel Corporation. DT was supported in part by the NSF grant CCF-1553428 and the NSF Frontier grant CNS-1413920. AI was supported in part by NSF awards CCF-1617730 and IIS-1741137. LE was supported in part by an MIT-IBM Watson AI Lab research grant. AM was supported in part by an Alfred P. Sloan Research Fellowship, a Google Research Award, and the NSF grants CCF-1553428 and CNS-1815221.

References

[1] Alexandre d'Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 19:1171–1183, 2008.

[2] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794, 2016.

[3] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *CoRR*, abs/1611.04231, 2016.

[4] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[5] Peter Henderson, Joshua Romoff, and Joelle Pineau. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*, 2018.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9:1–42, 1997.

[7] Sham M. Kakade. A natural policy gradient. In *NIPS*, 2001.

[8] Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[9] Kenji Kawaguchi. Deep learning without poor local minima. In *NIPS*, 2016.

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[12] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *NIPS*, 2014.

[13] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *CoRR*, abs/1803.07055, 2018.

[14] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[15] OpenAI, :, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2018.

[16] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, 2010.

[17] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham M. Kakade. Towards generalization and simplicity in continuous control. In *NIPS*, 2017.

[18] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *CoRR*, abs/1806.00451, 2018.

[19] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3): 400–407, 09 1951. doi: 10.1214/aoms/1177729586. URL `https://doi.org/10.1214/aoms/1177729586`.

[20] Itay Safran and Ohad Shamir. Spurious local minima are common in two-layer relu neural networks. In *ICML*, 2018.

[21] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[22] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. In *ICML*, 2015.

[23] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

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

[26] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[27] Ilya Sutskever. Keynote talk. NVIDIA NTECH, 2018. URL `https://www.youtube.com/watch?v=w3ues-NayAs&t=467s`.

[28] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[29] George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E. Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. In *ICML*, 2018.

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

9 Appendix

9.1 Experimental Setup

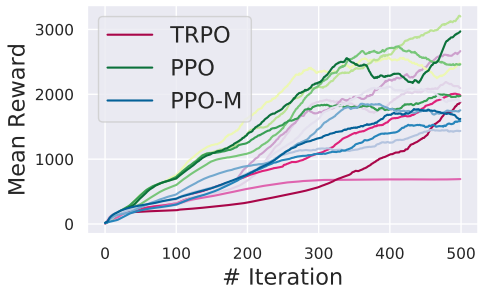
We use the following parameters for PPO, PPO-M, and TRPO based on a hyperparameter grid search:

Table 1: Hyperparameters for PPO and TRPO algorithms.

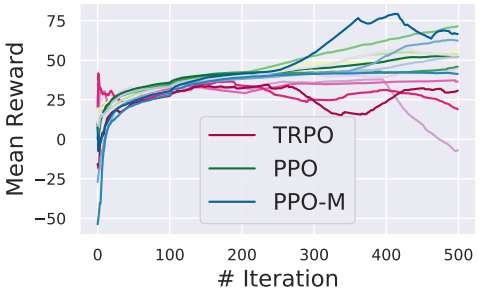
Hyperparameter	**Value**	
	TRPO	PPO
Timesteps per iteration	2000	
Discount factor (γ)	0.99	
GAE discount (λ)	0.95	
Value network LR	0.0001	
Value network num. epochs	10	
Policy network hidden layers	[64, 64]	
Value network hidden layers	[64, 64]	
Number of minibatches	N/A	32
Policy LR	N/A	0.0001
Policy epochs	N/A	10
Entropy coefficient	N/A	0.0
Clipping coefficient	N/A	0.2
KL constraint (δ)	0.01	N/A
Fisher estimation fraction	10%	N/A
Conjugate gradient steps	10	N/A
Conjugate gradient damping	0.1	N/A
Backtracking steps	10	N/A

All error bars we plot are 95% confidence intervals, obtained via bootstrapped sampling.

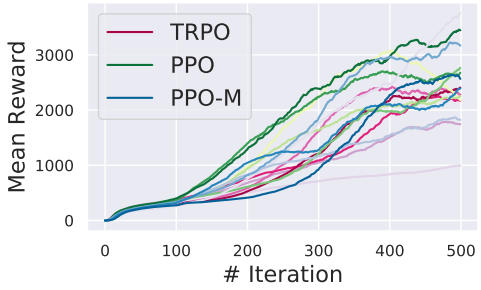
9.2 Standard Reward Plots



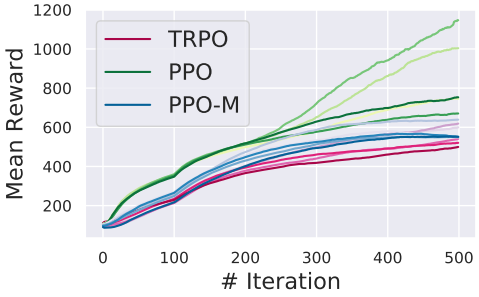
(a) Hopper-v2



(b) Swimmer-v2



(c) Walker2d-v2



(d) Humanoid-v2

Figure 10: Mean reward for the studied policy gradient algorithms on standard MuJoCo benchmark tasks. For each algorithm, we perform 5 random trials using the best performing hyperparameter configuration.

9.3 PPO Implementation Optimizations

Algorithm 1 PPO scaling optimization.

procedure INITIALIZE-SCALING()
    $R_0 \leftarrow 0$
    $RS =$ RUNNINGSTATISTICS() ▷ New running stats class that tracks mean, standard deviation
procedure SCALE-OBSERVATION(r_t) ▷ Input: a reward r_t
    $R_t \leftarrow \gamma R_{t-1} + r_t$ ▷ γ is the reward discount
    ADD(RS, R_t)
    return r_t/STANDARD-DEVIATION(RS) ▷ Returns scaled reward

9.3.1 Additional Optimizations

1. **Reward Clipping**: The implementation also clips the rewards within a preset range (usually $[-5, 5]$ or $[-10, 10]$).

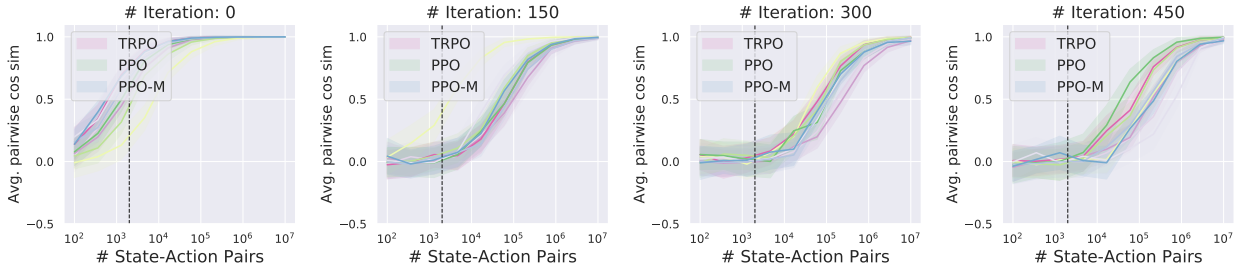
2. **Observation Normalization**: In a similar manner to the rewards, the raw states are also not fed into the optimizer. Instead, the states are first normalized to mean-zero, variance-one vectors.

3. **Observation Clipping**: Analagously to rewards, the observations are also clipped within a range, usually $[-10, 10]$.

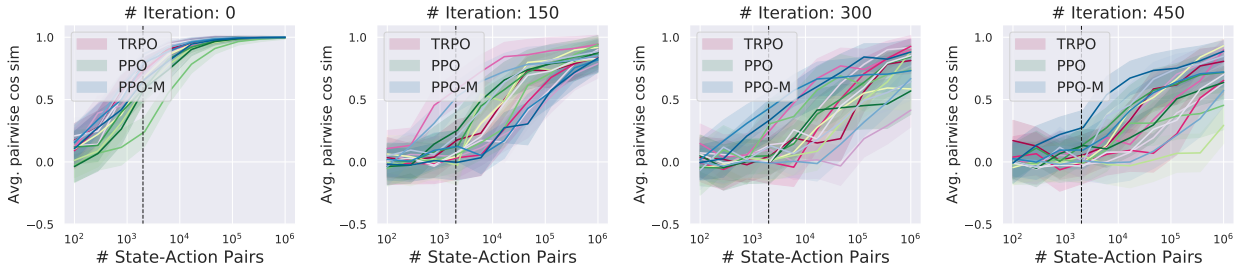
4. **Hyperbolic tan activations**: As also observed by [4], implementations of policy gradient algorithms also also use hyperbolic tangent function activations between layers in the policy and value networks.

5. **Global Gradient Clipping**: After computing the gradient with respect to the policy and the value networks, the implementation clips the gradients such the "global ℓ_2 norm" (i.e. the norm of the concatenated gradients of all parameters) does not exceed 0.5.

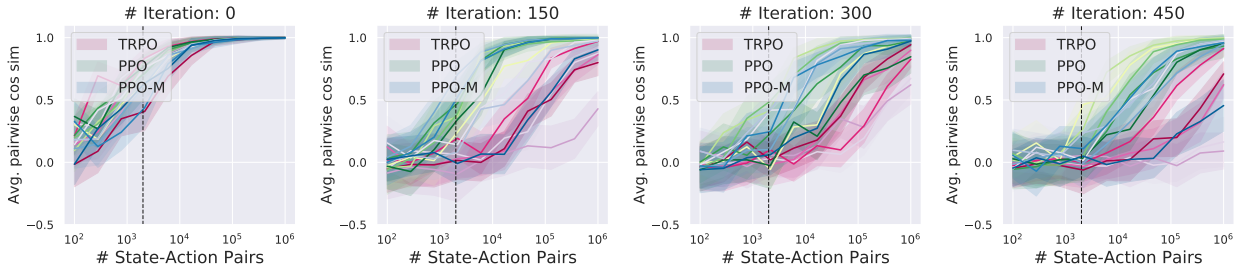
9.4 Quality of Gradient Estimation



(a) Walker2d-v2

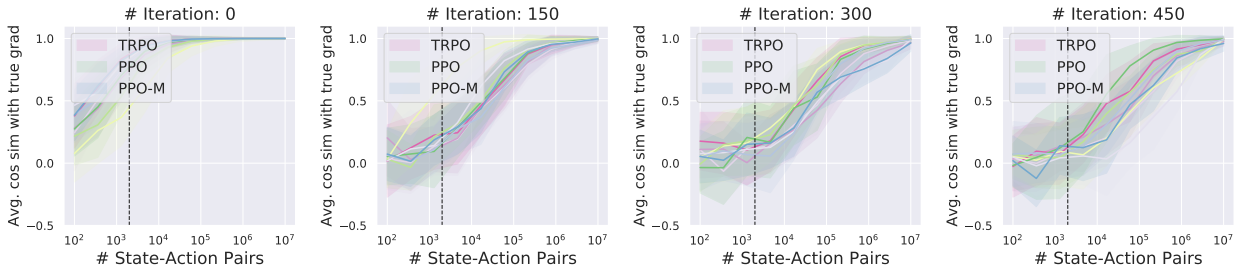


(b) Hopper-v2

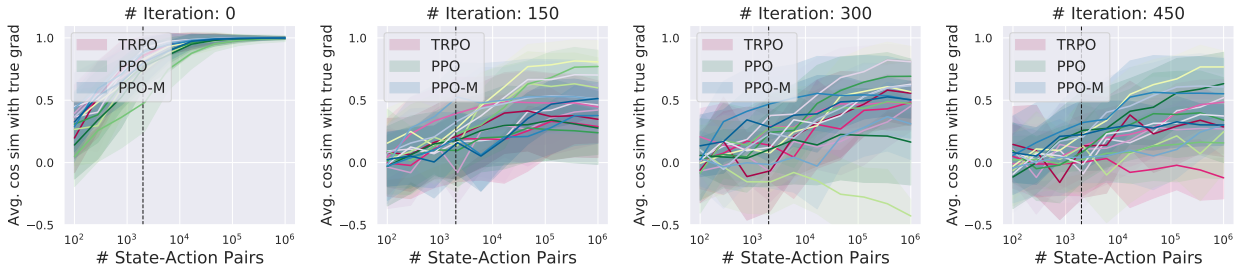


(c) Swimmer-v2

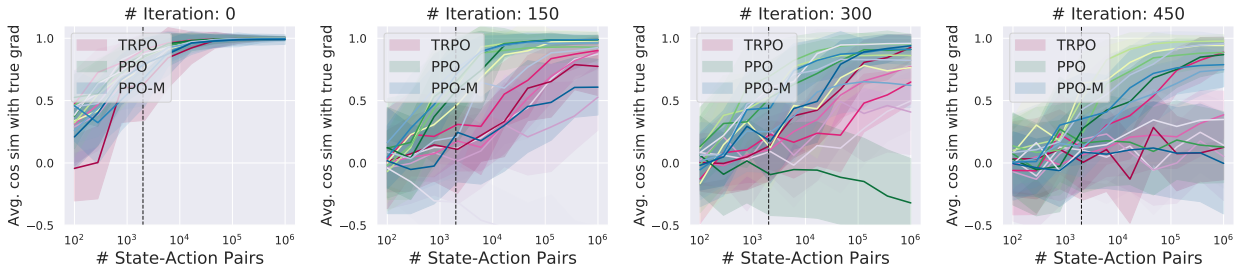
Figure 11: Empirical variance of the gradient (c.f. (15)) as a function of the number of state-action pairs used in estimation for policy gradient methods. We obtain multiple gradient estimates using a given number of state-action pairs from the policy at a particular iteration. We then measure the average pairwise cosine similarity between these repeated gradient measurements, along with the 95% confidence intervals (shaded). Each of the colored lines (for a specific algorithm) represents a particular trained agent (we perform multiple trials with the same hyperparameter configurations but different random seeds). The dotted vertical black line (at 2K) indicates the sample regime used for gradient estimation in standard practical implementations of policy gradient methods.



(a) Walker2d-v2

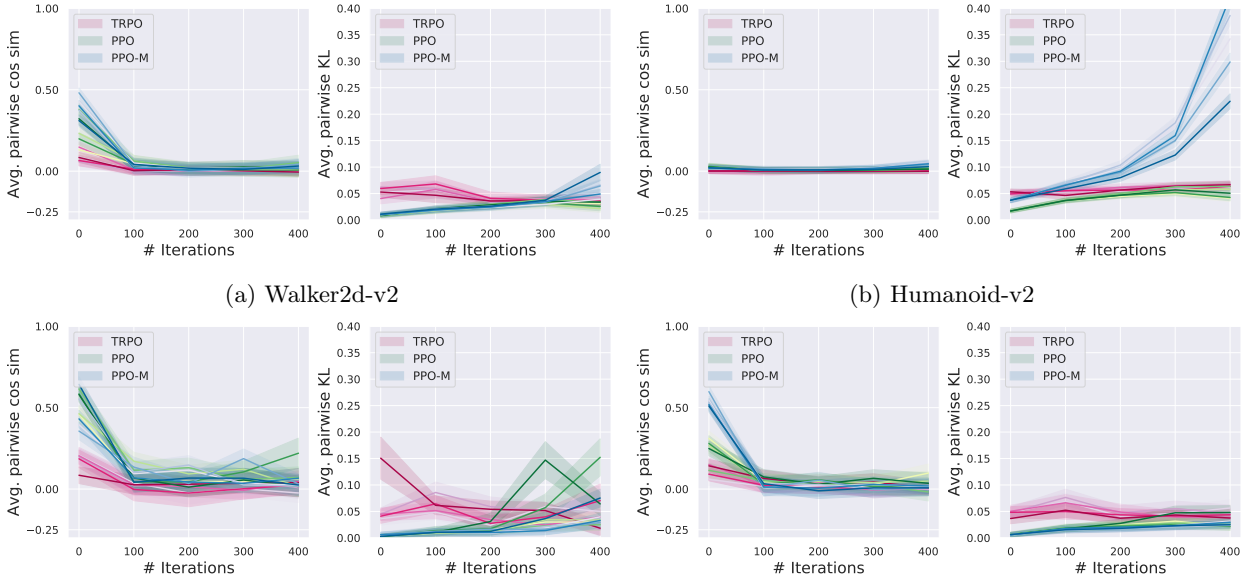


(b) Hopper-v2



(c) Swimmer-v2

Figure 12: Convergence of gradient estimates to the "true" expected gradient (c.f. (15)). We measure the cosine similarity between the true gradient (approximated using around 1M samples) and gradient estimates, as a function of number of state-action pairs used to obtain the later. For a particular policy and state-action pair count, we obtain multiple estimates of this cosine similarity and then report the average, along with the 95% confidence intervals (shaded). Each of the colored lines (for a specific algorithm) represents a particular trained agent (we perform multiple trials with the same hyperparameter configurations but different random seeds). The dotted vertical black line (at 2K) indicates the sample regime used for gradient estimation in standard practical implementations of policy gradient methods.



(a) Walker2d-v2

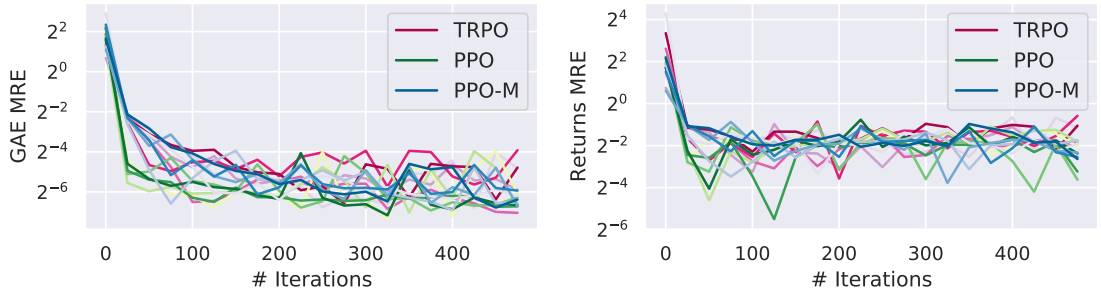
(b) Humanoid-v2

(c) Swimmer-v2

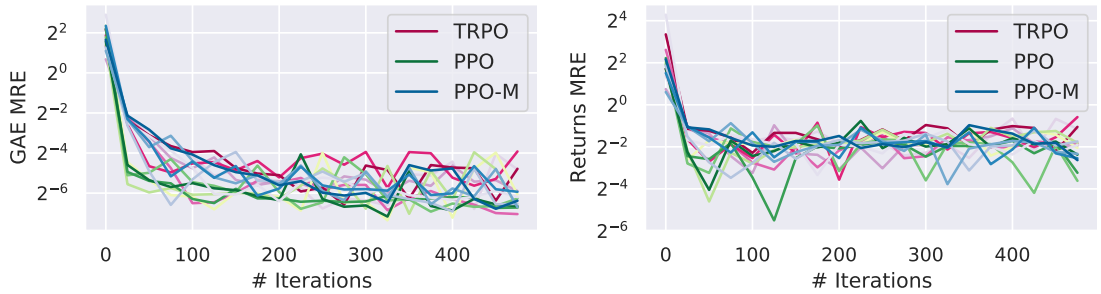
(d) Hopper-v2

Figure 13: Empirical variance of the policy update step over the course of training, by MuJoCo task we train on. For a given policy, we obtain one hundred estimates of the update step. We then measure the pairwise cosine similarity between these estimates and then report the average, along with the 95% confidence intervals (shaded). We also measure the pairwise KL divergence between the policy updates obtained using each of these steps. This seems to suggest that the policy steps exhibit high variance as well.

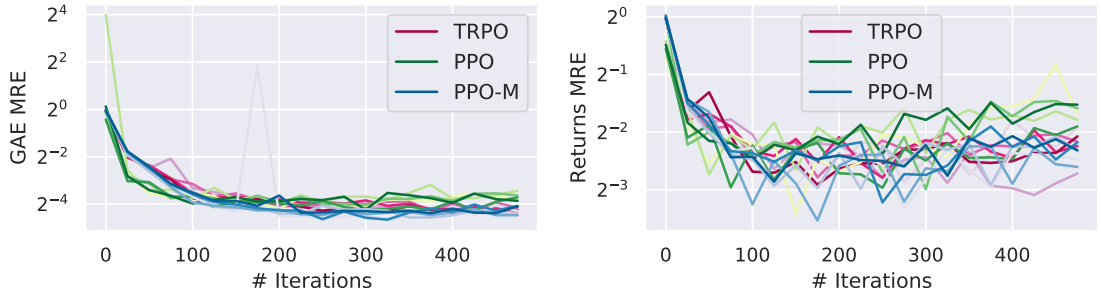
9.5 Value Prediction



(a) Hopper-v2

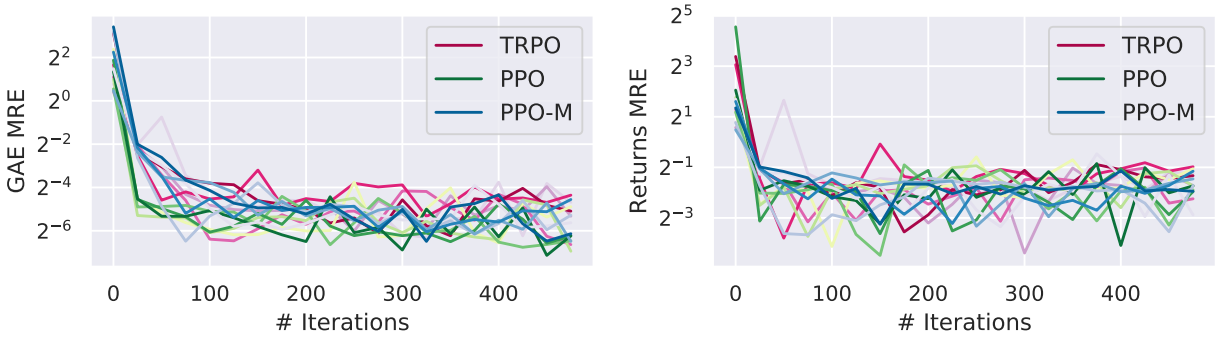


(b) Walker2d-v2

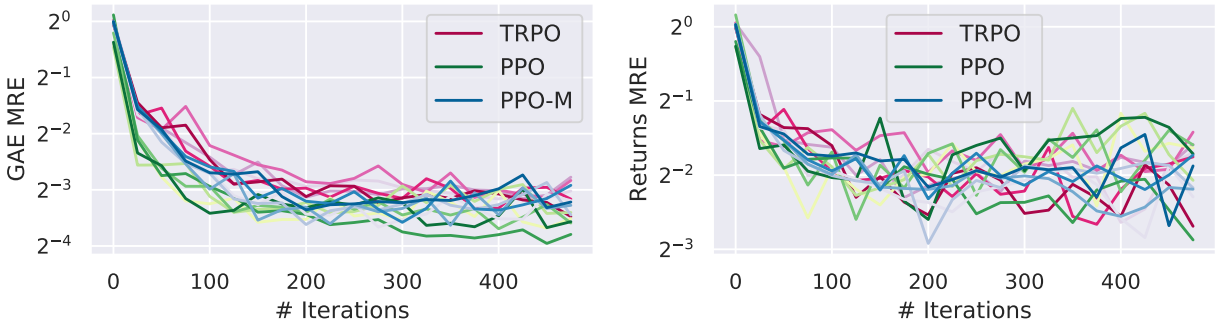


(c) Humanoid-v2

Figure 14: Quality of value prediction in terms of mean relative error (MRE) on train state-action pairs for agents trained to solve the MuJoCo tasks. We see in that the agents do indeed succeed at solving the supervised learning task they are trained for – the train MRE on the GAE-based value loss $(V_{old} + A_{GAE})^2$ (c.f. (17)) is small (left column). We observe that the returns MRE is quite small as well (right column).

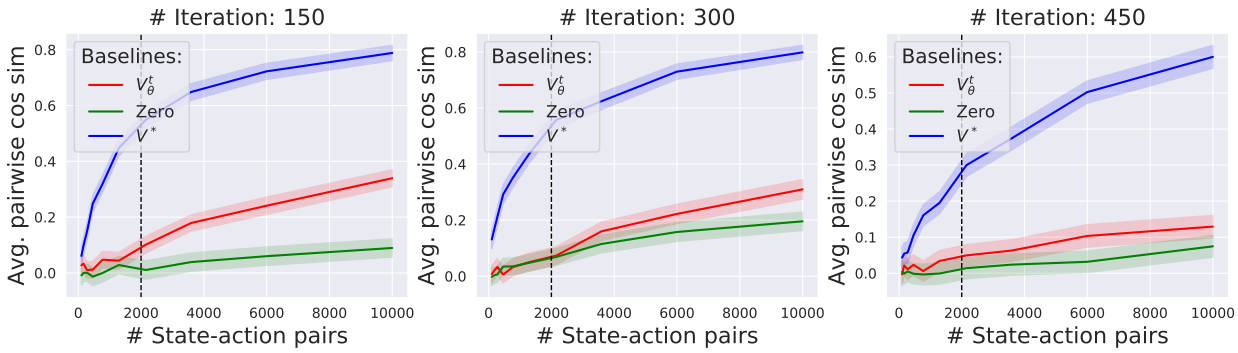


(a) Hopper-v2

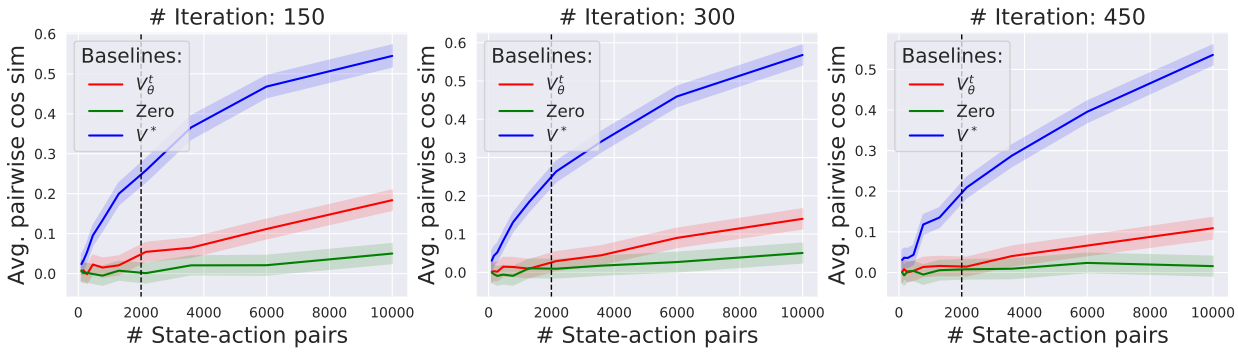


(b) Humanoid-v2

Figure 15: Quality of value prediction in terms of mean relative error (MRE) on heldout state-action pairs for agents trained to solve MuJoCo tasks. We see in that the agents do indeed succeed at solving the supervised learning task they are trained for – the validation MRE on the GAE-based value loss $(V_{old}+A_{GAE})^2$ (c.f. (17)) is small (left column). On the other hand, we see that the returns MRE is still quite high – the learned value function is off by about 50% with respect to the underlying true value function (right column).



(a) Hopper-v2



(b) Humanoid-v2

Figure 16: Efficacy of the value network as a variance reducing baseline function for agents trained on the Hopper and Walker2d-v2 MuJoCo task. We measure the empirical variance of the gradient (c.f. (15)) as a function of the number of state-action pairs used in estimation, for different choices of baseline functions. Specifically, we compare the baseline coming from: the value network (used by the agent during training), the "true" value function (fit directly to the returns using a large number of state-action pairs (500K) sampled from the *current* policy) and the "zero" value function (i.e. simply replacing advantages with returns). We observe that the usage of the true value function leads to a significantly lower-variance estimate of the gradient as compared to the value network. In turn, employing the value network yields a noticeably larger variance reduction compared to the zero baseline function, even though this difference may appear rather small in the most relevant, small-sample regime (2K).

9.6 Optimization Landscape

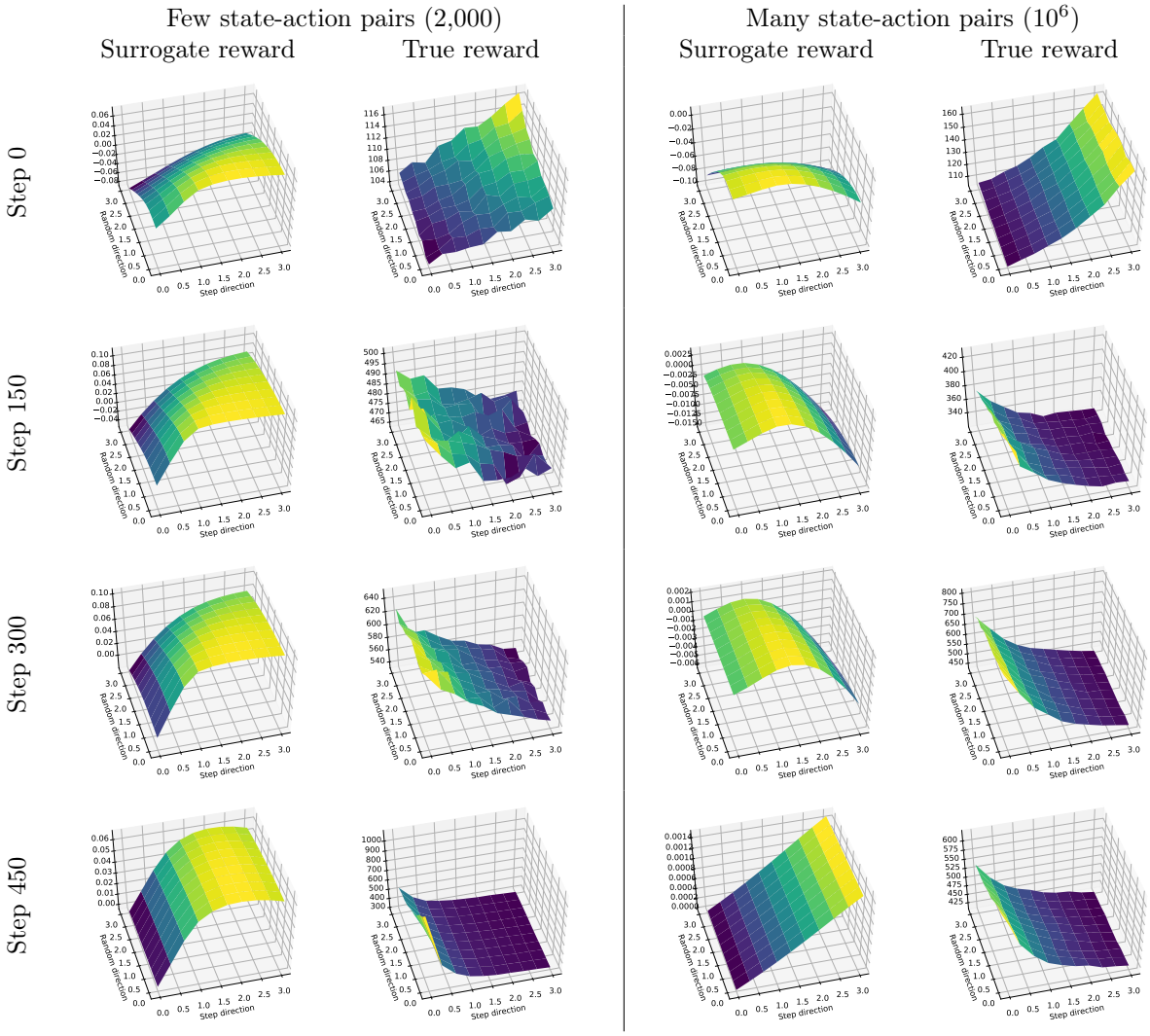


Figure 17: Humanoid-v2 – PPO reward landscapes.

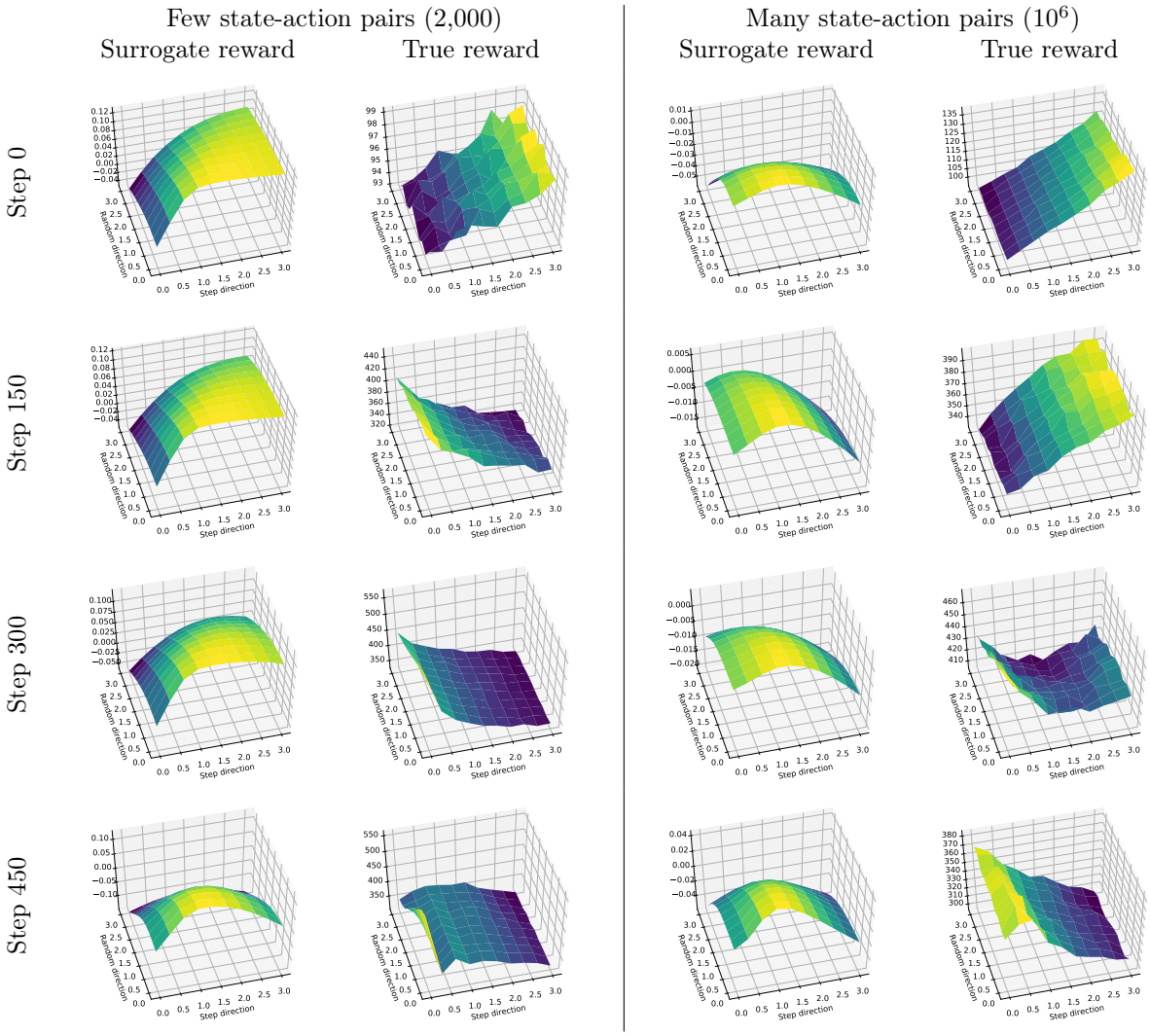


Figure 18: Humanoid-v2 – PPO-M reward landscapes.

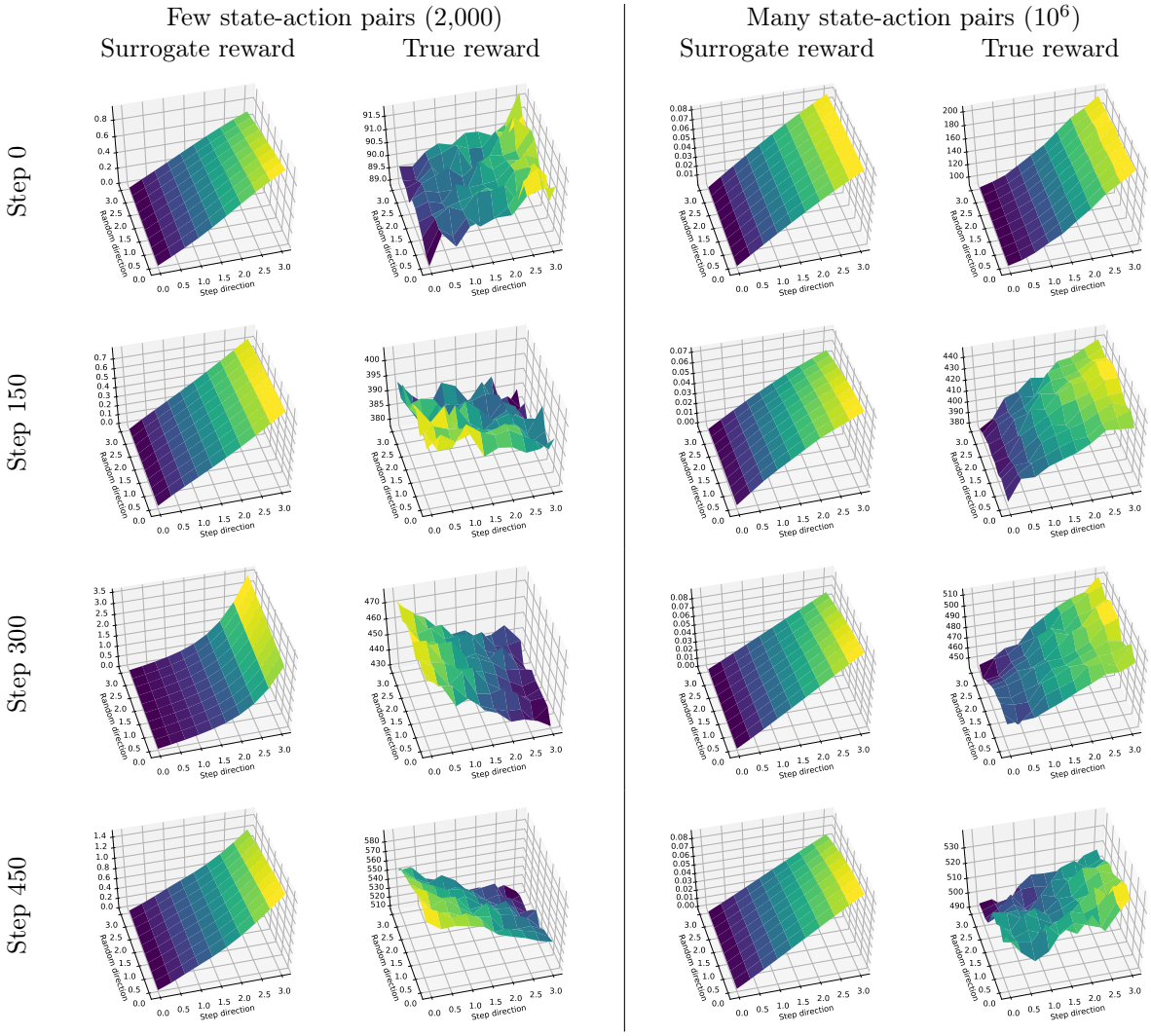


Figure 19: Humanoid-v2 – TRPO reward landscapes.

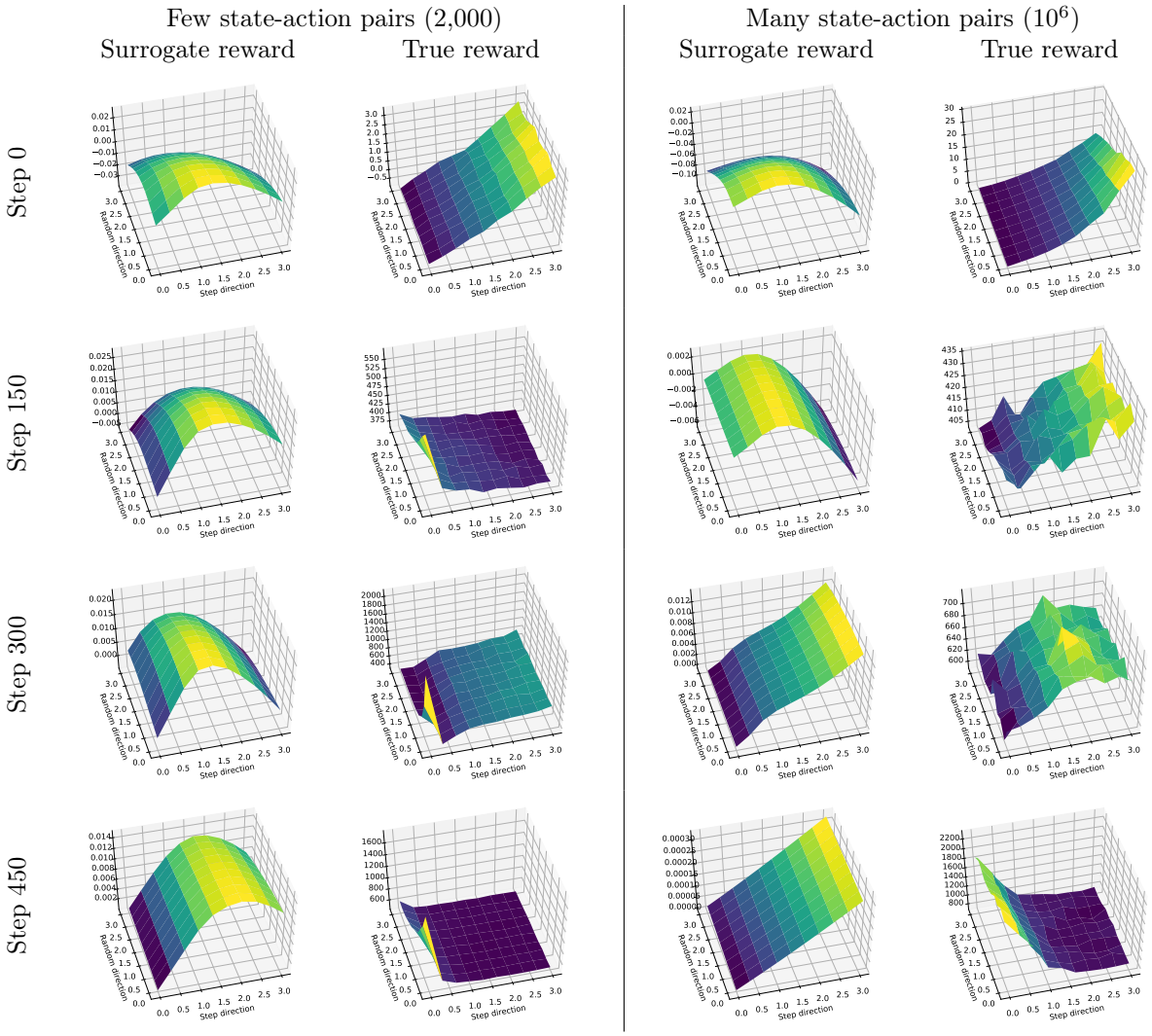


Figure 20: Walker2d-v2 – PPO reward landscapes.

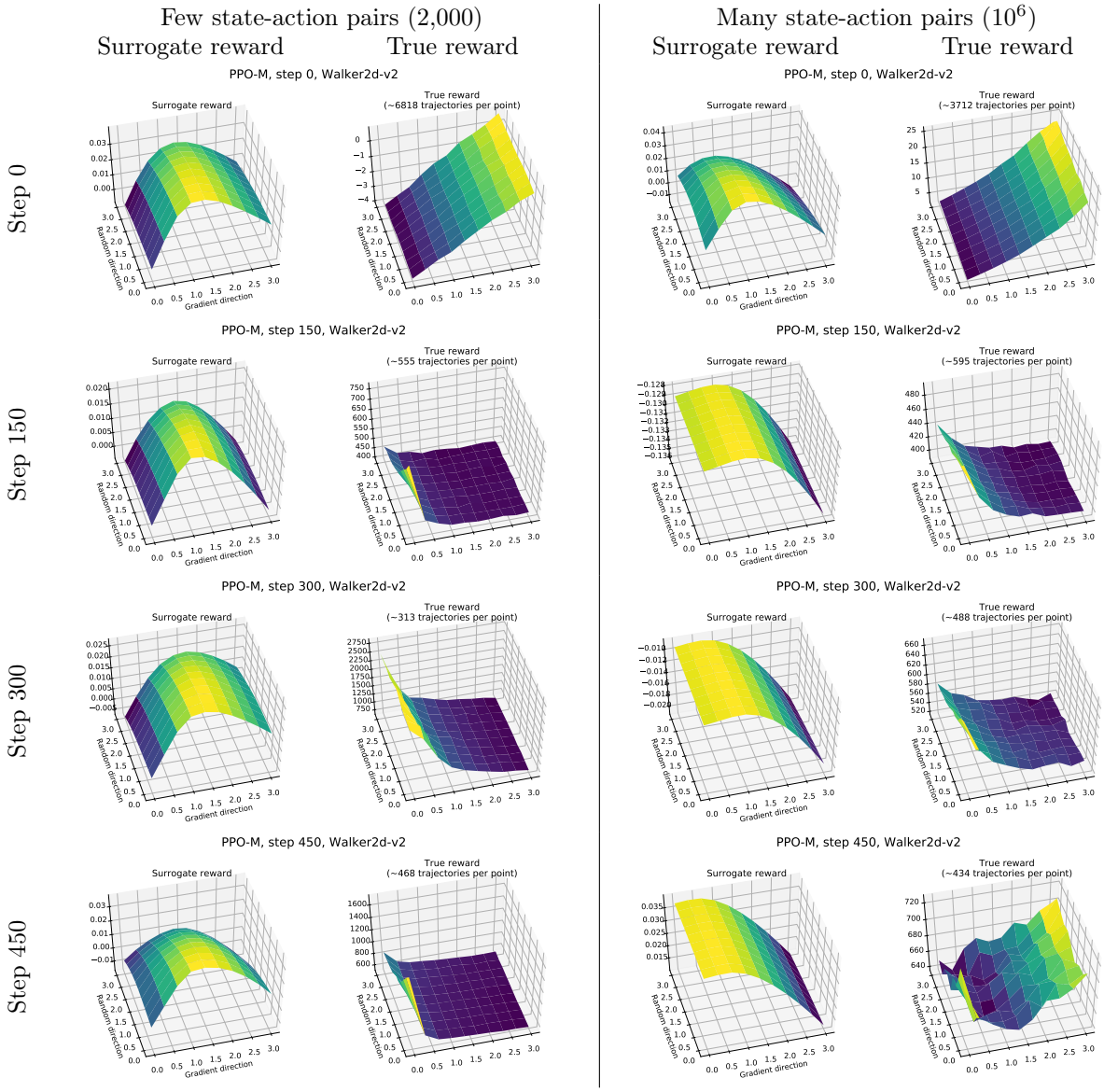


Figure 21: Walker2d-v2 – PPO-M reward landscapes.

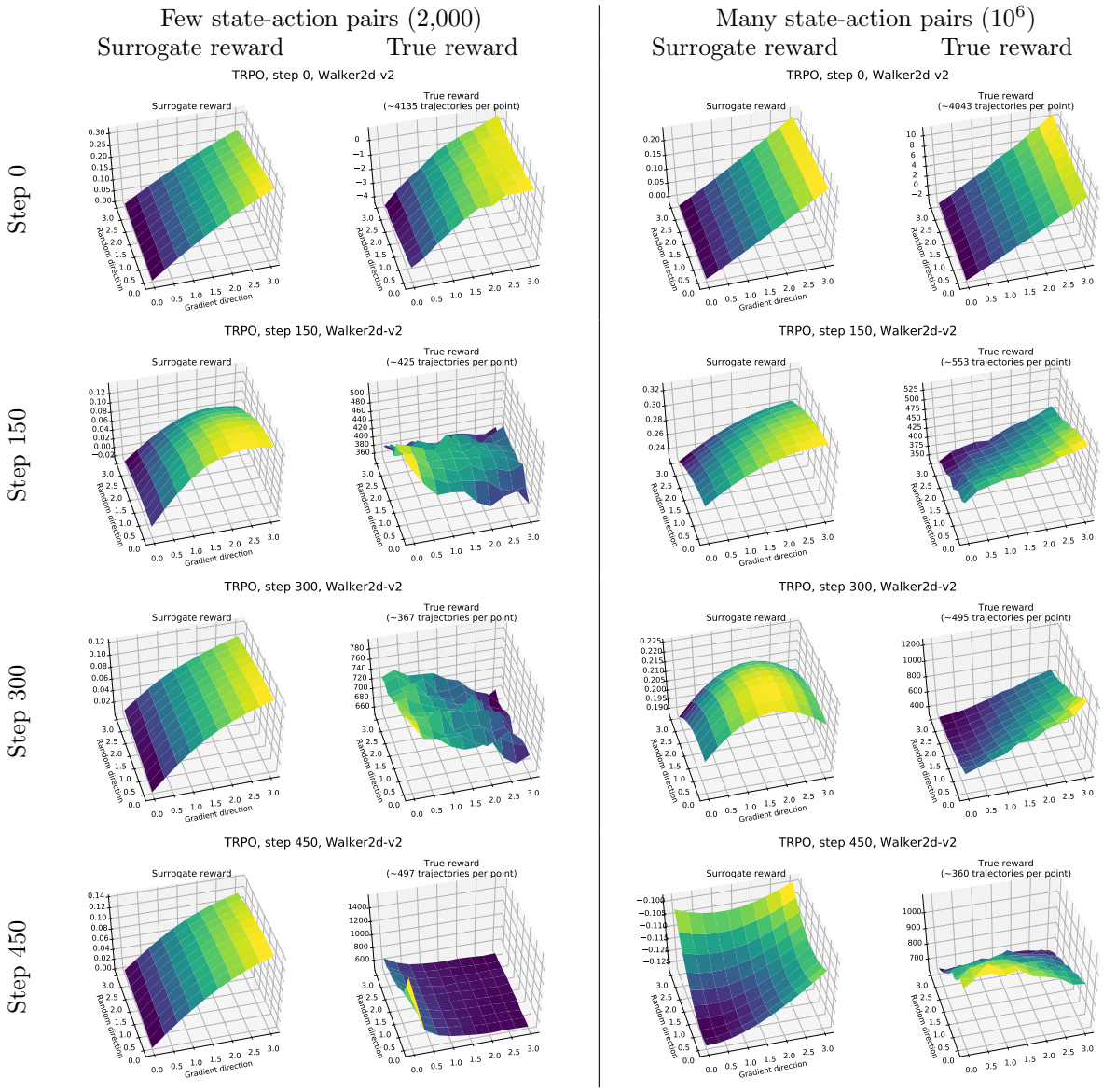


Figure 22: Walker2d-v2 – TRPO reward landscapes.

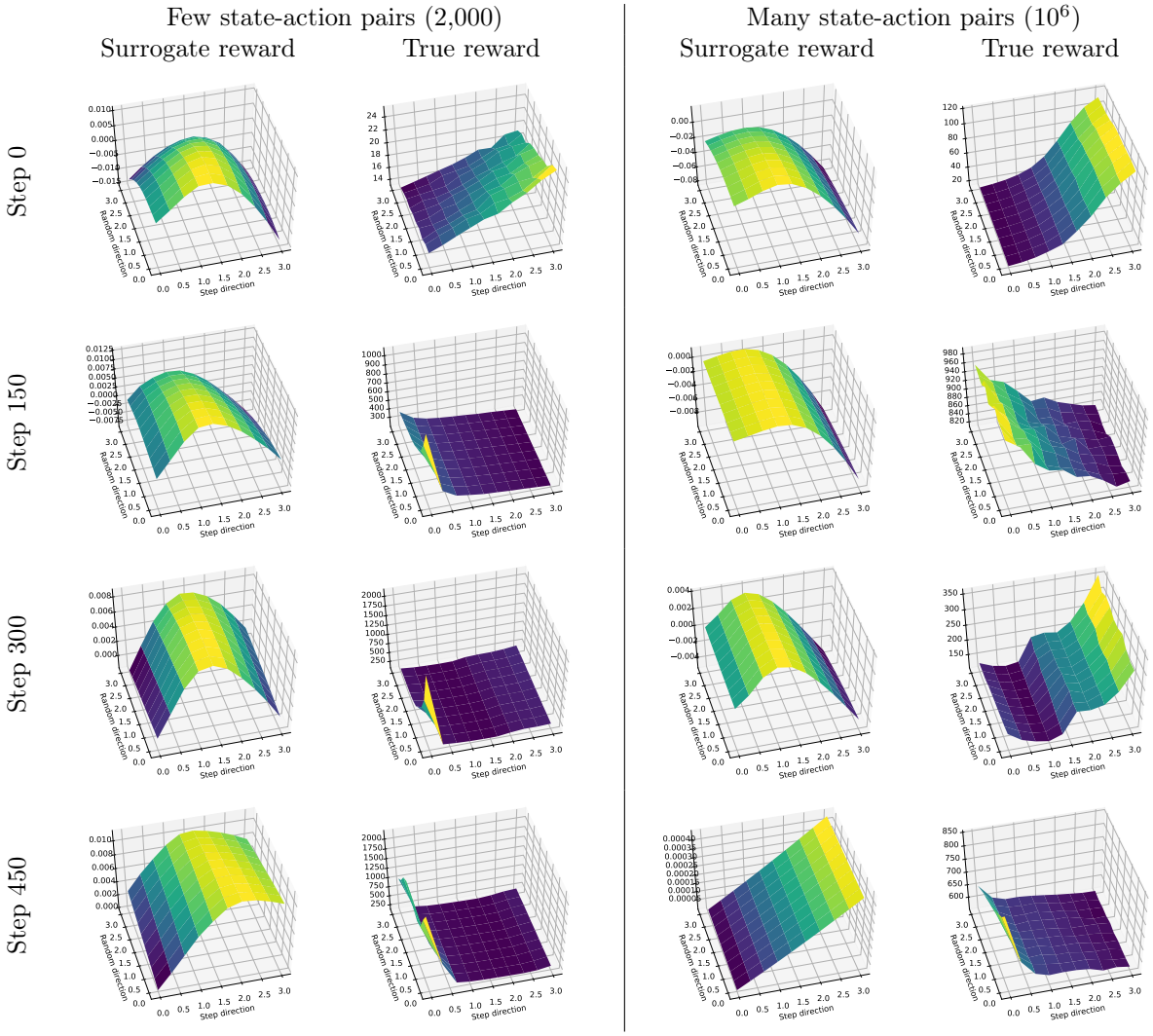


Figure 23: Hopper-v2 – PPO reward landscapes.

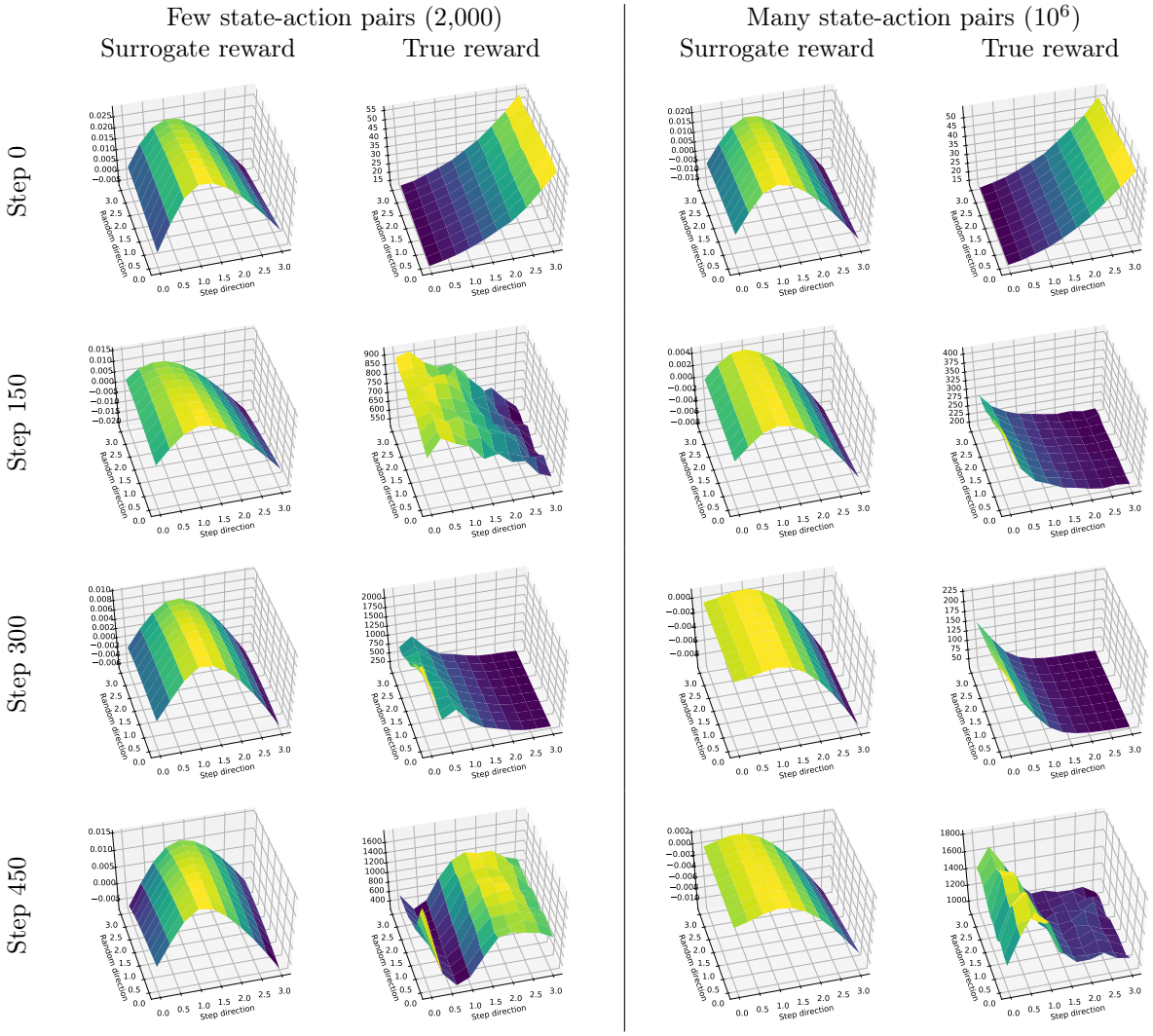


Figure 24: Hopper-v2 – PPO-M reward landscapes.

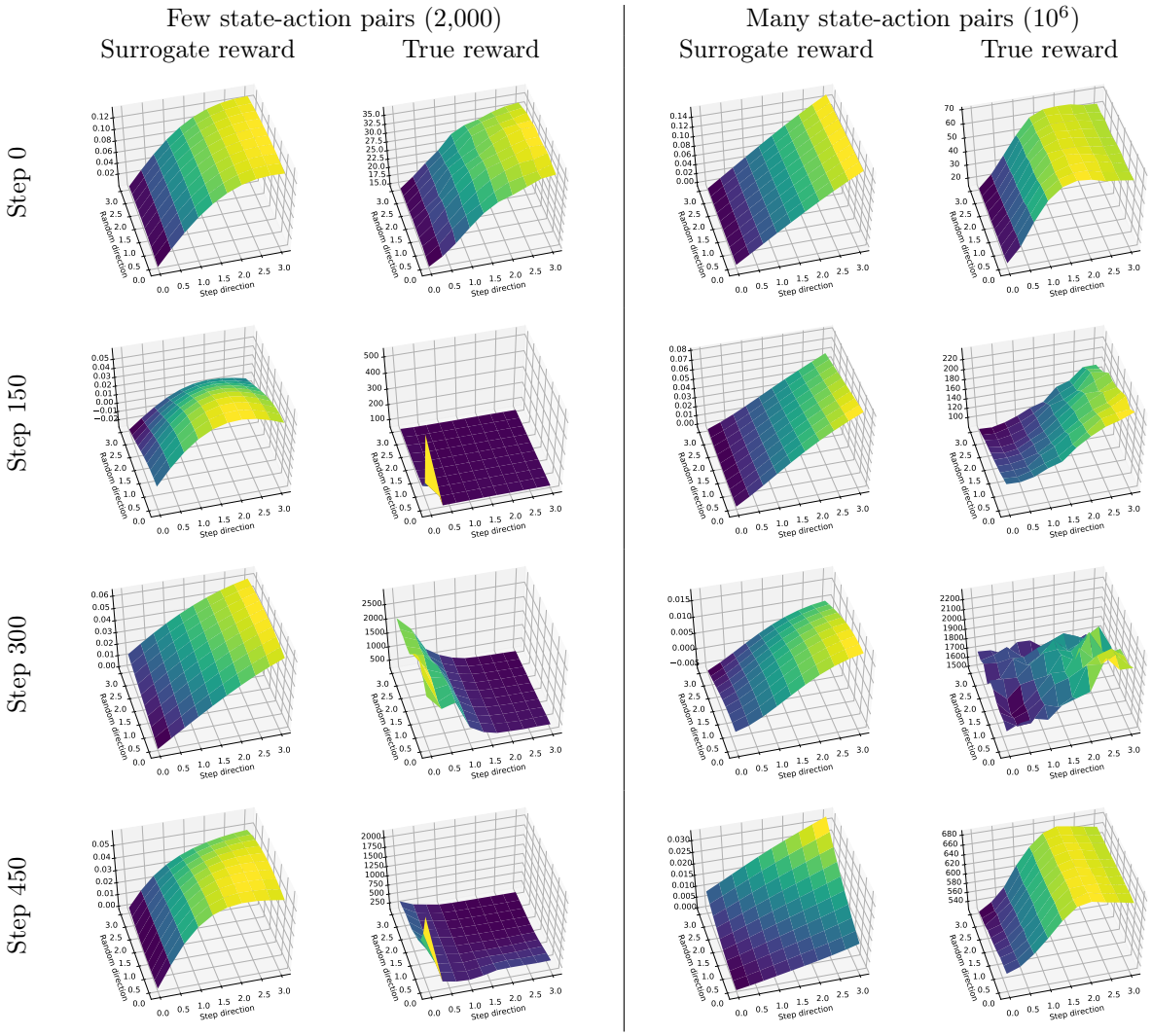


Figure 25: Hopper-v2 – TRPO reward landscapes.

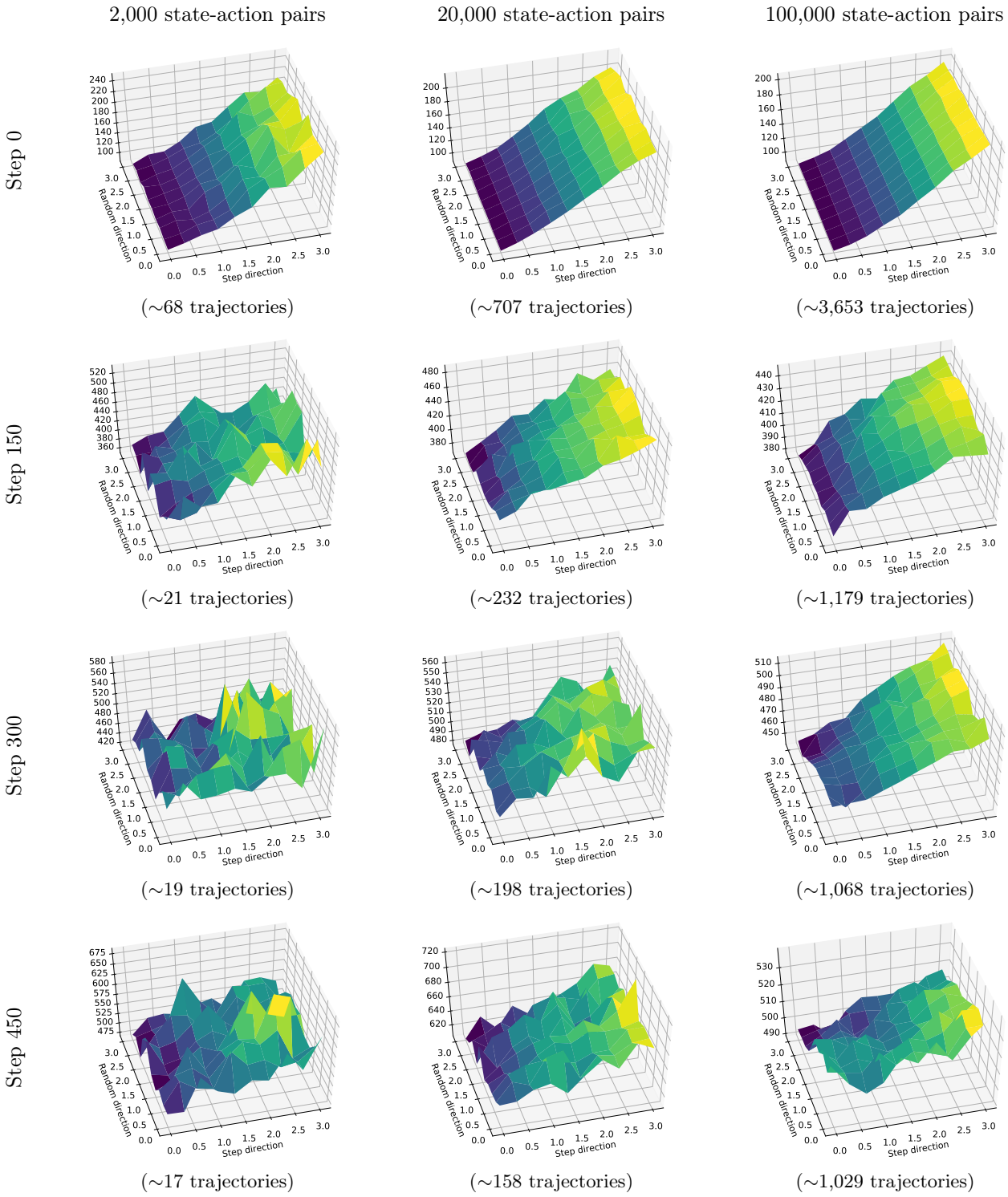


Figure 26: Humanoid-v2 TRPO landscape concentration (see Figure 6 for a description).

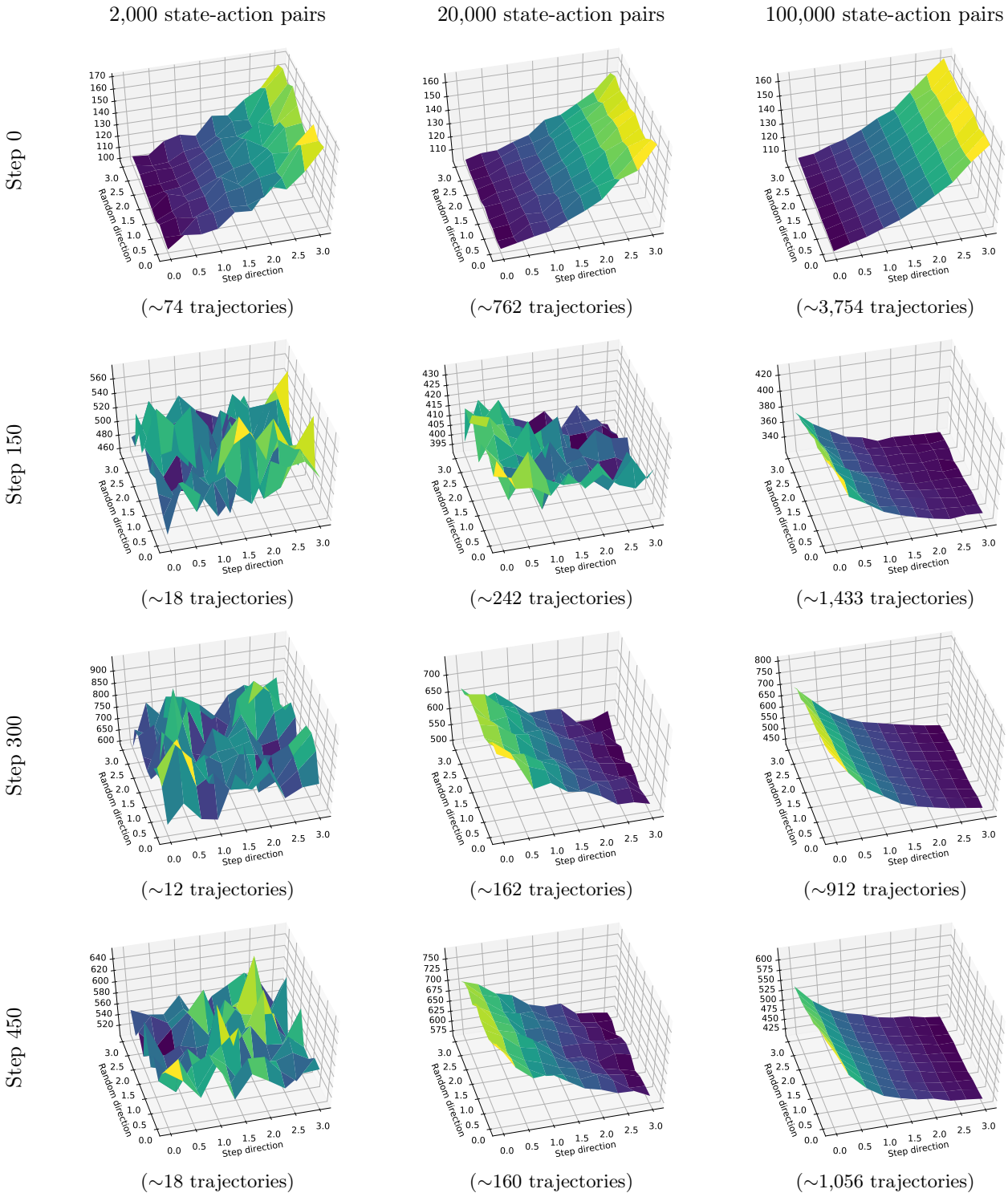
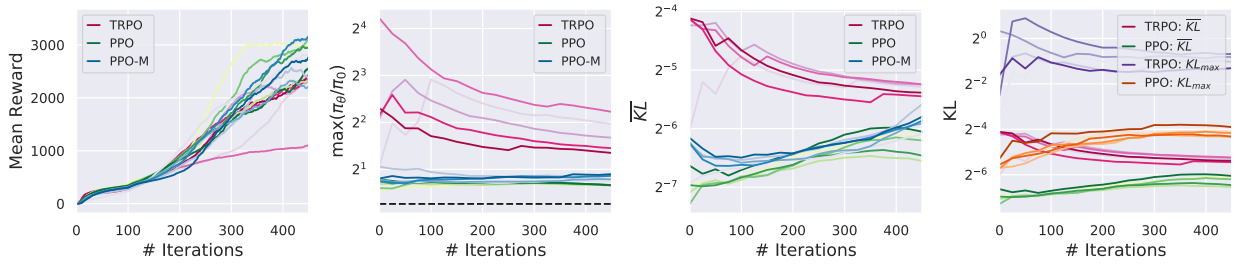
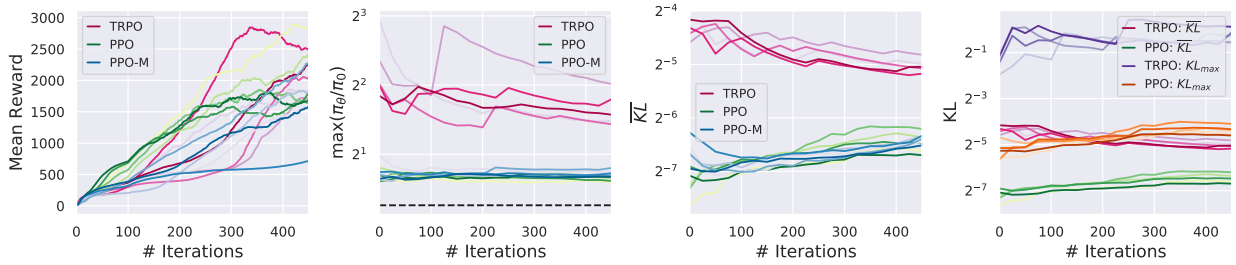


Figure 27: Humanoid-v2 PPO landscape concentration (see Figure 6 for a description).

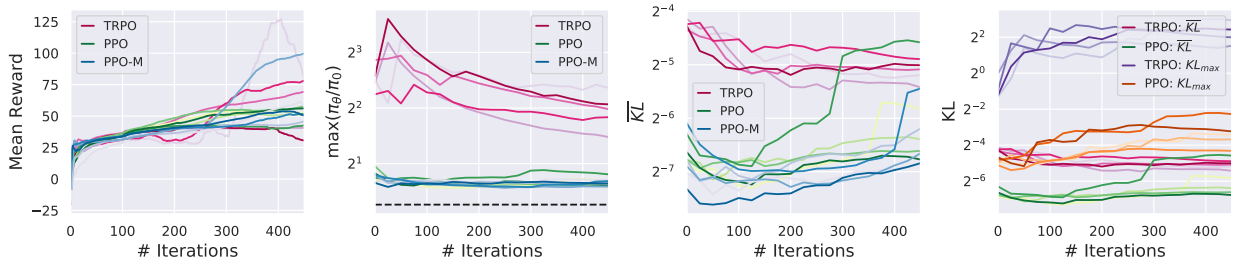
9.7 Trust Region Optimization



(a) Walker2d-v2 (train)

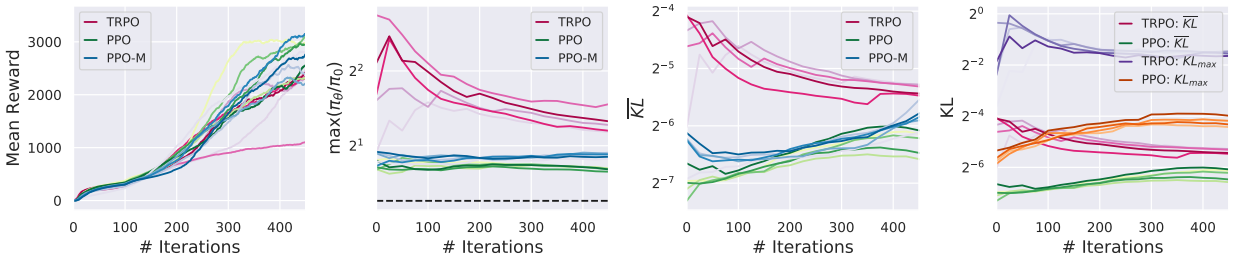


(b) Hopper-v2 (train)

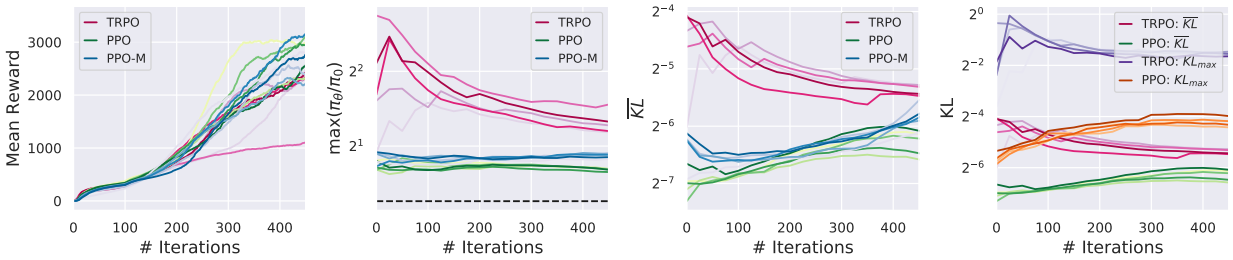


(c) Swimmer-v2 (train)

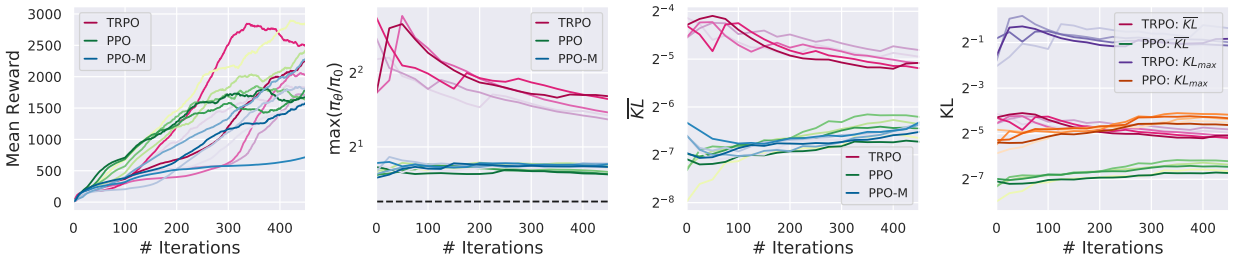
Figure 28: Per step mean reward, maximum ratio (c.f. (13)), mean KL, and maximum versus mean KL for agents trained to solve the MuJoCo Humanoid task. The quantities are measured over the state-action pairs collected in the *training step*. Each line represents a training curve from a separate agent. The black dotted line represents the $1 + \epsilon$ ratio constraint in the PPO algorithm, and we measure each quantity every twenty five steps. Compare the results here with Figure 29; they are qualitatively nearly identical.



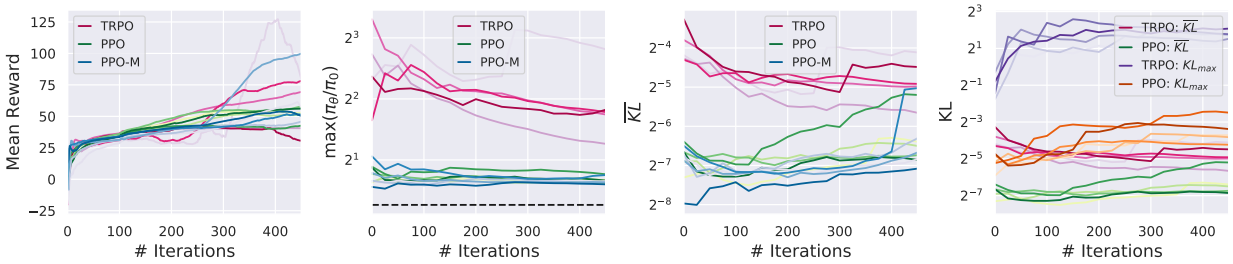
(a) Humanoid-v2 (heldout)



(b) Walker2d-v2 (heldout)



(c) Hopper-v2 (heldout)



(d) Swimmer-v2 (heldout)

Figure 29: Per step mean reward, maximum ratio (c.f. (13)), mean KL, and maximum versus mean KL for agents trained to solve the MuJoCo Humanoid task. The quantities are measured over state-action pairs collected from *heldout trajectories*. Each line represents a curve from a separate agent. The black dotted line represents the $1 + \epsilon$ ratio constraint in the PPO algorithm, and we measure each quantity every twenty five steps. See that the mean KL for TRPO nearly always stays within the desired mean KL trust region (at 0.06).

9.8 Proofs

Theorem 5.1 (Optima of the clipped objective; proof in Appendix 9.8)**.** *The optimization problem considered by proximal policy optimization (PPO), i.e.*

$$\min_{\{\pi_\theta(a|s)\}} \mathbb{E}_{(s,a)\in\tau\sim\pi}\left[clip\left(\frac{\pi_\theta(a|s)}{\pi(a|s)}, 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_\pi(s,a)\right],$$

either has (a) a unique solution that is identical to its unclipped counterpart, or (b) uncountably many optima, of which only one is within the trust region $[(1-\varepsilon)\pi, (1+\varepsilon)\pi]$.

Proof. Let θ^* denote an optimal solution of the *unclipped* surrogate objective, and θ_c^* be an optimum of the clipped objective. Now, suppose that the solution of the clipped problem above is not an optimum of its unclipped counterpart. This means that at least one of the ratios, $\pi_{\theta_c^*}(a|s)/\pi_0(a|s) \notin [1-\epsilon, 1+\epsilon]$. Otherwise θ_c^* would attain the same objective value in the clipped surrogate loss, and since the set of attainable values of the clipped surrogate loss are a subset of those of the unclipped surrogate loss, θ_c^* would thus be an optimum of the unclipped surrogate loss (violating our initial construction). Thus, we have that for at least one (a, s),

$$\pi_{\theta_c^*}(a|s)/\pi_0(a|s) \notin [1-\epsilon, 1+\epsilon].$$

Without loss of generality, suppose that $\pi_{\theta_c^*}(a|s) > (1+\varepsilon)\pi_\theta(a|s)$.

Note that $\text{clip}(x, 1-\varepsilon, 1+\varepsilon) = 1+\varepsilon$ for all $x \geq 1+\varepsilon$. Thus, any $\pi_\theta(a|s) > \pi_{\theta_c^*}$ leads to the same effective ratio $\pi_{\theta_c^*}/\pi_0$, which in turn leads to the same value of the objective. Thus, in probability space, this ratio can be set to one of uncountably many values without changing the objective value. Only one of these values $(\pi_{\theta_c^*}(a|s)/\pi_0(a|s) = 1+\epsilon)$ actually obeys the trust region. □