IMPLEMENTATION MATTERS IN DEEP POLICY GRADIENTS: A CASE STUDY ON PPO AND TRPO

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the roots of algorithmic progress in deep policy gradient algorithms through a case study on two popular algorithms, Proximal Policy Optimization and Trust Region Policy Optimization. We investigate the consequences of "code-level optimizations:" algorithm augmentations found only in implementations or described as auxiliary details to the core algorithm. Seemingly of secondary importance, such optimizations have a major impact on agent behavior. Our results show that they (a) are responsible for most of PPO's gain in cumulative reward over TRPO, and (b) fundamentally change how RL methods function. These insights show the difficulty, and importance, of attributing performance gains in deep reinforcement learning.

1 INTRODUCTION

Deep reinforcement learning (RL) algorithms have fueled many of the most publicized achievements in modern machine learning (Silver et al., 2017; OpenAI, 2018; Abbeel & Schulman, 2016; Mnih et al., 2013). However, despite these accomplishments, deep RL methods still are not nearly as reliable as their (deep) supervised counterparts. Indeed, recent research found the existing deep RL methods to be brittle (Henderson et al., 2017; Zhang et al., 2018), hard to reproduce (Henderson et al., 2017; Tucker et al., 2018), unreliable across runs (Henderson et al., 2017; 2018), and sometimes outperformed by simple baselines (Mania et al., 2018).

The prevalence of these issues points to a broader problem: we do not understand how the parts comprising deep RL algorithms impact agent training, either separately or as a whole. This unsatisfactory understanding suggests that we should re-evaluate the inner workings of our algorithms. Indeed, the overall question motivating our work is: how do the multitude of mechanisms used in deep RL training algorithms impact agent behavior?

Our contributions. We will specifically analyze the underpinnings of agent behavior—both cumulative reward, as well as more fine-grained algorithmic properties. As a first step towards tackling this question, we conduct a case study of two of the most popular deep policy-gradient methods: Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) and Proximal Policy Optimization (PPO) (Schulman et al., 2017). These algorithms are closely related: PPO was originally developed as a refinement of TRPO.

We find that much of the method's performance comes from various code-level or implementation optimizations not present in TRPO: seemingly small modifications to the core algorithm either found only in a paper's original implementation, or described as auxiliary details. [1] We pinpoint these optimizations, and run an ablation study demonstrating that they are instrumental to the PPO's performance.

This observations prompt us to study how such code-level optimizations change agent training dynamics, and whether we can truly think of them as merely auxiliary improvements. Our results indicate that code-level optimizations fundamentally change algorithms' operation, going beyond improvements in agent reward. Concretely, we find that these optimizations are in fact essential

[1] Note that these code-level optimizations are separate from "implementation choices" like the choice of Pytorch versus TensorFlow in that they intentionally change the training algorithm's operation.

for satisfying a key motivating principle behind TRPO and PPO's operations: trust region enforcement. Additionally, we find that these optimizations are both necessary and sufficient to maintain a trust region, *regardless* of whether or not the clipping algorithm—typically thought to be the central algorithm of PPO—is employed.

Ultimately, we find that the *PPO code-optimizations are significantly more important in terms of final reward achieved* than the choice of general training algorithm (TRPO vs. PPO). This result is in stark contrast to the previous view that the central PPO clipping method drives the gains seen in Schulman et al. (2017). In doing so, we demonstrate that the algorithmic changes imposed by such optimizations make rigorous comparisons of algorithms difficult. Without a rigorous understanding of the full impact of code-level optimizations, we cannot hope to gain any reliable insight from comparing algorithms on benchmark tasks.

Our results emphasize the importance of building RL methods in a modular manner. To progress towards more performant and reliable algorithms, we need to understand each component's impact on agent behavior and performance—both individually, and as part of a whole.

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 (1992), who proposed the REINFORCE algorithm. Later, Sutton et al. (1999) established a unifying framework that casts the previous algorithms as instances of the policy gradient method.

Our work focuses on proximal policy optimization (PPO) (Schulman et al., 2017) and trust region policy optimization (TRPO) (Schulman et al., 2015a), 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 (2001). 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. (2010), which constrains the distance between marginal action distributions (whereas TRPO constrains the conditionals of such action distributions).

Notably, Henderson et al. (2017) points out a number of brittleness, reproducibility, and experimental practice issues in deep RL algorithms. Importantly, we build on the observation of Henderson et al. (2017) that final reward for a given algorithm is greatly influenced depending on the code base used. Rajeswaran et al. (2017) and Mania et al. (2018) 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. (2018) 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 Attributing Success in Proximal Policy Optimization

Our overarching goal is to better understand the underpinnings of the behavior of deep policy gradient methods. We thus perform a careful study of two tightly linked algorithms: TRPO and PPO (recall that PPO is motivated as TRPO with a different trust region enforcement mechanism). To better understand these methods, we start by thoroughly investigating their implementations in practice. We find that in comparison to TRPO, the PPO implementation contains many non-trivial optimizations that are not (or only barely) described in its corresponding paper. Indeed, the standard implementation of PPO [2] contains the following additional optimizations (among many others; we provide a full list in Appendix A.2):

1. **Value function clipping:** Schulman et al. (2017) originally suggest fitting the value network via regression to target values:

$$L^V = (V_{\theta_t} - V_{targ})^2,$$

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

but in the standard implementation the value network is instead fit with a PPO-like objective:

$$L^V = \min\left[\left(V_{\theta_t} - V_{targ}\right)^2, \left(\text{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{targ}\right)^2\right],$$

where V_θ is clipped around the previous value estimates (and ε is fixed to the same value as the value used in (2) to clip the probability ratios).

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 A.2.
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 (Kingma & Ba, 2014) (an already adaptive method) for optimization.

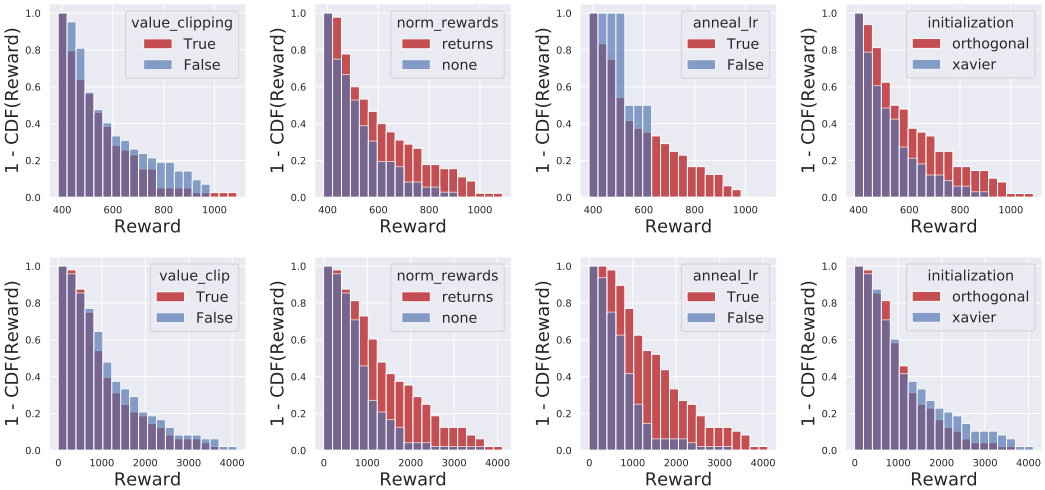


Figure 1: An ablation study on the four optimizations described in Section 3 (value clipping, reward scaling, network initialization, and learning rate annealing). For each of the 2^4 possible configurations of optimizations, we train a Humanoid-v2 (top) and Walker2d-v2 (bottom) agent using PPO with 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. We detail our experimental setup in Appendix A.1.

These optimizations may appear as merely surface-level or insignificant algorithmic changes to the core policy gradient method at hand. However, we find that they dramatically affect the performance of PPO. To demonstrate this, we start by performing 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.

The above findings show that our ability to understand PPO from an algorithmic perspective hinges on the ability to distill out its fundamental principles from such algorithm-independent (in the sense

[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 A.2

that these optimizations can be implemented for any policy gradient method) optimizations. We thus 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. PPO-M also ignores all the code-level optimizations listed in Appendix A.2: reward clipping, observation normalization, observation clipping, hyperbolic tan activations, and global gradient clipping. We then explore PPO-M alongside PPO and TRPO.

Overall, our results on the importance of these optimizations both corroborate results demonstrating the brittleness of deep policy gradient methods, and demonstrate that even beyond environmental brittleness, the algorithms themselves exhibit high sensitivity to implementation choices[4].

4 CODE-LEVEL OPTIMIZATIONS HAVE ALGORITHMIC EFFECTS

In the previous section, we found that canonical implementations of PPO contain many *code-level optimizations*: implementation choices that are not motivated as core to a method but profoundly impact performance.

This mismatch leads us to ask: *how do these seemingly superficial code-level optimizations impact underlying agent behavior?* In this section, we demonstrate that the code-level optimizations fundamentally alter agent behavior. Rather than merely improving ultimate cumulative award, such optimizations directly impact the principles motivating the core algorithms.

Trust Region Optimization. A key property of policy gradient algorithms is that update steps computed at any specific policy π_{θ_t} are only guaranteed predictiveness in a neighborhood around θ_t. Thus, to ensure that the update steps we derive remain predictive many policy gradient algorithms ensure that these steps stay in the vicinity of the current policy. The resulting "trust region" methods (Kakade, 2001; Schulman et al., 2015a; 2017) 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) Schulman et al. (2015a). TRPO constrains the KL divergence between successive policies on the optimization trajectory, leading to the following problem:

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}\widehat{A}_\pi(s_t,a_t)\right] \\ \text{s.t.} \quad & D_{KL}(\pi_\theta(\cdot \mid s)||\pi(\cdot \mid s)) \leq \delta, \quad \forall s \ . \end{aligned} \tag{1}$$

In practice, we maximize this objective with a second-order approximation of the KL divergence and natural gradient descent, and replace 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.

One disadvantage of the TRPO algorithm is that it 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. (2017) propose proximal policy optimization (PPO), which tries to enforce a trust region with a different objective that does not require computing a projection. Concretely, PPO proposes replacing the KL-constrained objective (1) of TRPO by clipping the objective function directly as:

$$\max_{\theta} \mathbb{E}_{(s_t,a_t)\sim\pi}\left[\min\left(\text{clip}\left(\rho_t, 1-\varepsilon, 1+\varepsilon\right)\widehat{A}_\pi(s_t,a_t),\ \rho_t\widehat{A}_\pi(s_t,a_t)\right)\right] \tag{2}$$

where

$$\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)} \tag{3}$$

In addition to its simplicity, PPO is intended to be faster and more sample-efficient than TRPO (Schulman et al., 2017).

[4]This might also explain the difference between different codebases observed in Henderson et al. (2017)

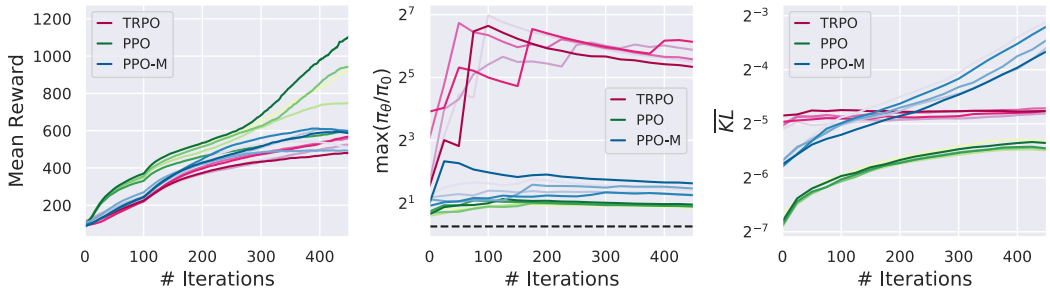


Figure 2: Per step mean reward, maximum ratio (c.f. (2)), mean KL, and maximum versus mean KL for agents trained to solve the MuJoCo Humanoid task. The quantities are measured over the state-action pairs collected in the *training step*. Each line represents a training curve from a separate agent. The black dotted line represents the $1 + \epsilon$ ratio constraint in the PPO algorithm, and we measure each quantity every twenty five steps. We take mean and max KL over the KL divergences between the conditional distributions induced by the current and previous policy on the observed states in training (at each step). In the left plot we see the reward for each trained agent. From in the middle plot, we can see that the PPO variants' maximum ratios consistently violates the ratio "trust region." In the right plot, we 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 measure the quantities over a *heldout* set of state-action pairs and find little qualitative difference in the results (seen in Figure 5 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 4. We detail our experimental setup in Appendix A.1.

Trust regions in TRPO and PPO. Enforcing a trust region is a core *algorithmic* property of different policy gradient methods. However, whether or not a trust region is enforced is not directly observable from the final rewards. So, how does this algorithmic property vary across state-of-the-art policy gradient methods?

In Figure 2 we measure the mean KL divergence between successive policies in a training run of both TRPO and PPO-M (PPO without code-level optimizations). Recall that TRPO is designed specifically to constrain this KL-based trust region, while the clipping mechanism of PPO attempts to approximate it. Indeed, while TRPO precisely enforcing this trust region, the successive KL divergence between policies in PPO-M grows *exponentially* as training progresses.

While this may seem surprising at first, we find that the unbounded nature of the trust region actually follows naturally from the clipping mechanism of PPO. In particular, the contribution of a single state-action pair to the gradient of the PPO objective is given by:

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

$$\text{where} \qquad L_\theta := \mathbb{E}_{(s,a)\in\tau\sim\pi}\left[\frac{\pi_\theta(a|s)}{\pi(a|s)} A_\pi(s,a)\right],$$

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

are respectively the standard and clipped versions of the surrogate objective. 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 objective.

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. In fact, observe in Figure 2 that PPO-M fails at even maintaining a trust region based on the maximum ratio (i.e., the exact quantity that PPO tries to control via clipping).

Remarkably, despite that the core mechanism of PPO (which is captured in PPO-M) fails to maintain a trust region, we find that PPO with optimizations actually *does* seem to maintain a KL-based trust

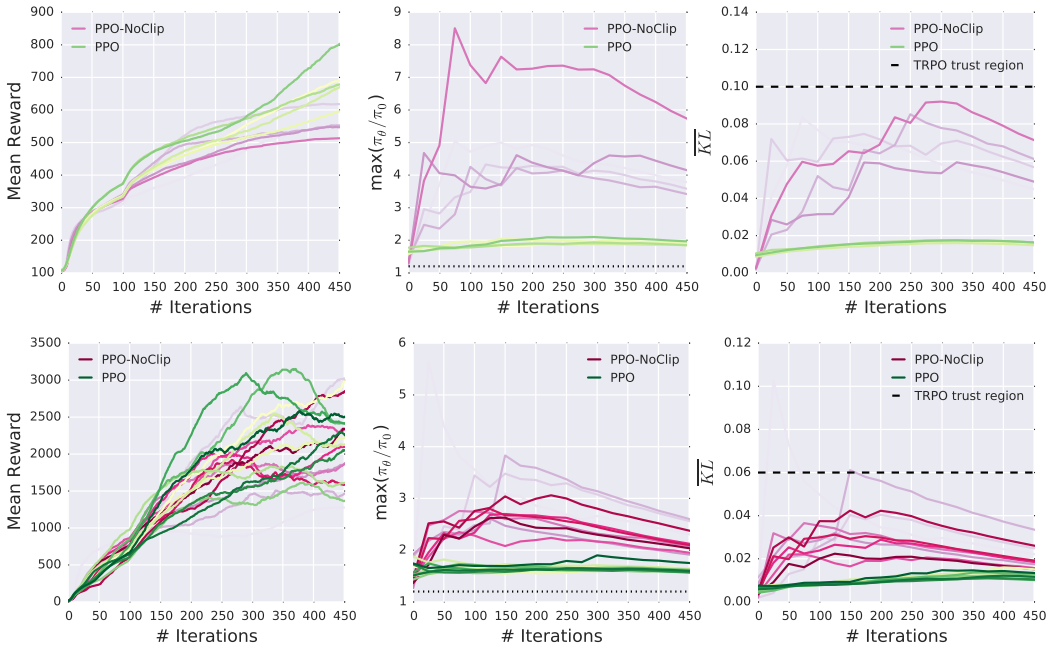


Figure 3: Per step mean reward, maximum ratio (c.f. (2)), mean KL, and mean KL for PPO and PPO-NoClip agents trained to solve the MuJoCo Humanoid-v2 (top) and Hopper-v2 (bottom) task. The quantities are measured over the state-action pairs collected in *heldout steps*: i.e., these state-action pairs were sampled independently of those used to construct the steps. Each line represents a training curve from a separate agent. PPO-NoClip represents the PPO algorithm, with code-level optimizations, but without any clipping. The left plot indicates the cumulative reward of each agent. The middle plot shows that the agent violates the maximum ratio. However, the right plot indicates that the agents maintain a bounded trust region (indeed, more bounded than the trust region of the best TRPO agent, shown as the black line on the right), and achieve high reward, despite not using the PPO clipping step at all. We give the same plots corresponding to other benchmark tasks in Appendix 6—in the same way, the clipping-free agents often succeed at maintaining a very reasonable trust region without using the core clipping mechanism. As previously, we detail our experimental setup in Appendix A.1.

region. This demonstrates that perhaps the key to PPO's success even from an algorithmic viewpoint comes from auxiliary optimizations, rather than the core methodology.

Enforcing a trust region without projecting or clipping. It turns out that code-level optimizations *alone* enforce a trust region without clipping the objective function *or* explicitly bounding the distance between successive policies. Indeed, Figure 3 demonstrates that PPO-NoClip (PPO without clipping), when accompanied by the original PPO code-level optimizations, can actually maintain reasonable trust regions on benchmark tasks. The trust region for PPO-NoClip bounds KL more tightly than the TRPO KL bound (this is represented by the horizontal, black dotted line in the mean KL plot), and we do not observe the same exponentially increasing trend we found in PPO-M in Figure 2.

Overall, our results indicate that so-called code-level optimizations do not merely increase performance: they fundamentally change algorithms' operation in unexpected ways.

5 Identifying Roots of Algorithmic Progress

State-of-the-art deep policy gradient methods are comprised of many interacting components. At what is generally described as their core, these methods incorporate mechanisms like trust region-enforcing steps, time-dependent value predictors, and advantage estimation methods for controlling

the exploitation/exploration trade-off (Schulman et al., 2015b). However, these algorithms also incorporate many less oft-discussed optimizations (cf. Section 3) that ultimately dictate much of agent behavior (cf. Section 4). Given the need to improve on these algorithms, the fact that such optimizations are so important begs the question: *how do we identify the true roots of algorithmic progress in deep policy gradient methods?*

Unfortunately, we find that answering this question is not easy. Going back to our study of PPO and TRPO, it is widely believed (and claimed) that the key innovation of PPO responsible for its improved performance over the baseline of TRPO is the ratio clipping mechanism (cf. Section 4). However, we have already shown that this clipping mechanism does not enforce the KL region it is supposed to. Where is PPO's improved performance coming from, then?

To address this question, we set out to disentangle the impact of PPO's core clipping mechanism from its code-level optimizations. Specifically, we examine how employing the core PPO and TRPO steps changes model performance while controlling for the effect of code-level optimizations identified in standard implementations of PPO (cf. Section 3). (Note that these code-level optimizations are algorithm independent: they can be straightforwardly applied to any policy gradient method.) To account for the effects of these optimizations, we study an additional algorithm which we denote as *TRPO+*, consisting of the core algorithmic contribution of TRPO in combination with PPO's code-level optimizations as identified in Section 3. We note that the four algorithms we study (PPO, PPO-M, TRPO, and TRPO+) now capture all combinations of core algorithms and code-level optimizations, allowing us to study the impact of each in a more fine-grained manner.

As our results show in Table 1, it turns out that code-level optimizations contribute to algorithms' increased performance often *significantly more* than the choice of algorithm (i.e., using PPO vs. TRPO). For example, on Walker2d-v2, PPO and TRPO see 28% and 39% improvements (respectively) when equipped with code-level optimizations. At the same time, after fixing the choice to use or not use optimizations, employing the core TRPO algorithm yields improvements of 20% and 10% (for the cases of with and without-optimizations respectively). In fact, our results suggest that much of PPO's improved performance (over TRPO) actually stems from code-level optimizations—on three of the four tasks, TRPO+ actually outperforms PPO.

Given the relative insignificance of the step mechanism compared to the use of code-level optimizations, we are prompted to ask: *to what extent is the clipping mechanism of PPO even necessary?* In Table 2, we assess this by considering a PPO-NoClip algorithm which uses code-level optimizations but no clipping mechanism (this is the same algorithm we studied in Section 4 in the context of trust region enforcement). It turns out that the clipping mechanism is not at all necessary to achieve high performance— we find that PPO-NoClip achieves similar results to PPO in terms of benchmark performance without any clipping at all.

Table 1: Full ablation of step choices (PPO or TRPO) and presence of code-level optimizations measuring agent performance on benchmark tasks. TRPO+ is a variant of TRPO that uses PPO inspired code-level optimizations, and PPO-M is a variant of PPO that does not use PPO's code-level optimizations (cf. Section 3). Surprisingly, we find that varying the use of code-level optimizations impacts performance significantly more than varying whether the PPO or TRPO step is used. We detail our experimental setup in Appendix A.1. We train 10 agents for each estimate.

Task	PPO	PPO-M	TRPO	TRPO+
Swimmer-v2	58 ± 6	58 ± 7	31 ± 15	94 ± 10
Hopper-v2	2175 ± 431	1816 ± 248	2009 ± 332	2245 ± 243
Walker2d-v2	2769 ± 250	2160 ± 435	2381 ± 458	3309 ± 286
Humanoid-v2	939 ± 109	558 ± 26	564 ± 25	638 ± 27

Our results suggest that it is difficult to attribute success to different aspects of policy gradient algorithms without careful analysis.

Table 2: Comparison of PPO performance to PPO without clipping. We find that there is little difference between the rewards attained between the two algorithms on the benchmark tasks. Note that both algorithms use code-level optimizations; our results indicate that the clipping mechanism is not very important compared to the use of code-level optimizations. We detail our experimental setup in Appendix A.1. We train 10 agents for each estimate.

TASK	PPO	PPO-M	PPO-NOCLIP
SWIMMER-V2	58 ± 6	58 ± 7	56 ± 15
HOPPER-V2	2175 ± 431	1816 ± 248	2467 ± 275
WALKER2D-V2	2769 ± 250	2160 ± 435	2692 ± 446
HUMANOID-V2	939 ± 109	558 ± 26	913 ± 164

6 CONCLUSION

In this work, we take a first step in examining how the mechanisms powering deep policy gradient methods impact agents both in terms of achieved reward and underlying algorithmic behavior. Wanting to understand agent operation from the ground up, we take a deep dive into the operation of two of the most popular deep policy gradient methods: TRPO and PPO. In doing so, we identify a number of "code-level optimizations"—algorithm augmentations found only in algorithms' implementations or described as auxiliary details in their presentation—and find that these optimizations have a drastic effect on agent performance.

In fact, these seemingly unimportant optimizations fundamentally change algorithm operation in ways unpredicted by the conceptual policy gradient framework. Indeed, the optimizations prove necessary for enforcing trust regions *regardless* of whether we optimize with the PPO step or just unconstrained stochastic gradient descent. We go on to test the importance of these code-level optimizations in agent performance, and find that PPO's marked improvement over TRPO (and even stochastic gradient descent) is largely due to these optimizations.

Overall, our results highlight the necessity of designing deep RL methods in a *modular* manner. When building algorithms, we should understand precisely how each component impacts agent training—both in terms of overall performance and underlying algorithmic behavior. It is impossible to properly attribute successes and failures in the complicated systems that make up deep RL methods without such diligence. More broadly, our findings suggest that developing an RL toolkit will require moving beyond the current benchmark-driven evaluation model to a more fine-grained understanding of deep RL methods.

REFERENCES

Pieter Abbeel and John Schulman. Deep reinforcement learning through policy optimization. *Tutorial at Neural Information Processing Systems*, 2016.

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

Peter Henderson, Joshua Romoff, and Joelle Pineau. Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods, 2018.

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

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

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NeurIPS Deep Learning Workshop*, 2013.

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

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

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

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

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

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

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.

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

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.

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

Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning, 2018.

A APPENDIX

A.1 EXPERIMENTAL SETUP

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

Table 3: 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.

A.2 PPO CODE-LEVEL OPTIMIZATIONS

Algorithm 1 PPO scaling optimization.

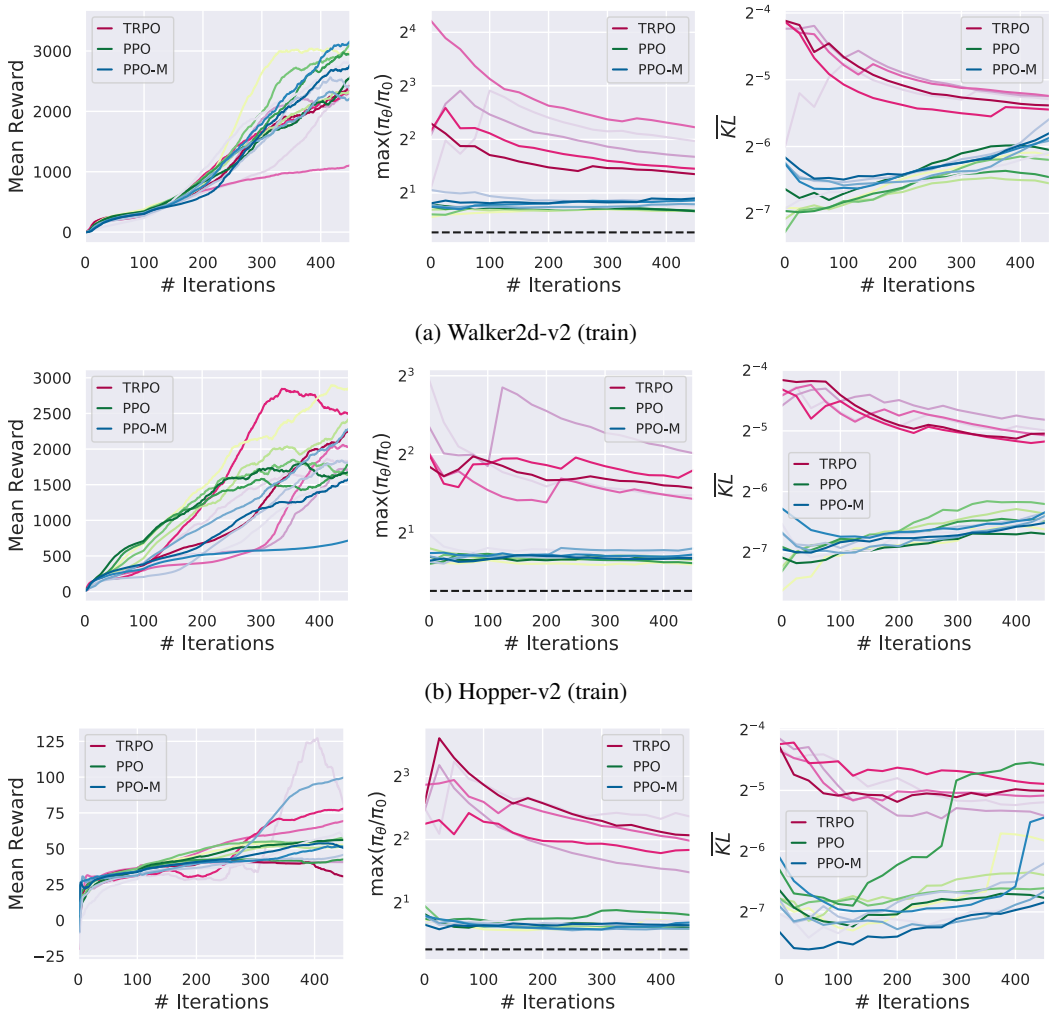
procedure INITIALIZE-SCALING()
  $R_0 \leftarrow 0$
  $RS =$ RUNNINGSTATISTICS() ▷ New running stats class that tracks mean, standard deviation
procedure SCALE-OBSERVATION(r_t) ▷ Input: a reward r_t
  $R_t \leftarrow \gamma R_{t-1} + r_t$ ▷ γ is the reward discount
  ADD(RS, R_t)
  return r_t/STANDARD-DEVIATION(RS)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

A.2.1 ADDITIONAL OPTIMIZATIONS

In addition to the optimizations listed in Section 3, PPO also uses the following 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 Henderson et al. (2017), 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.

A.3 TRUST REGION OPTIMIZATION

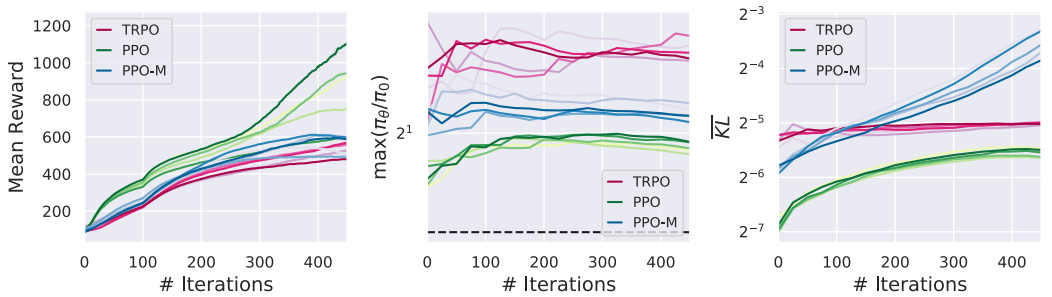


(a) Walker2d-v2 (train)

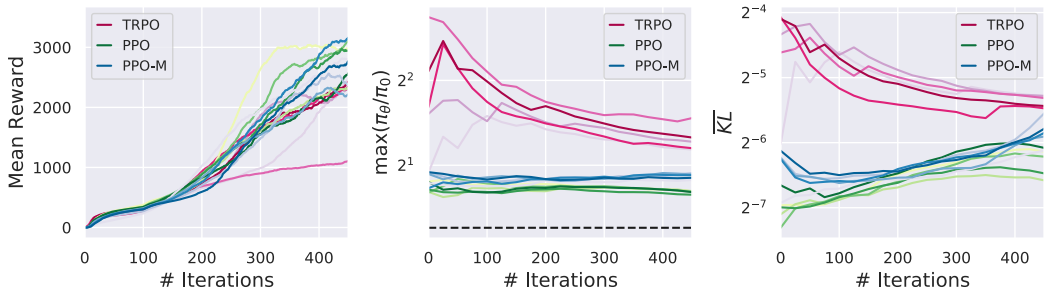
(b) Hopper-v2 (train)

(c) Swimmer-v2 (train)

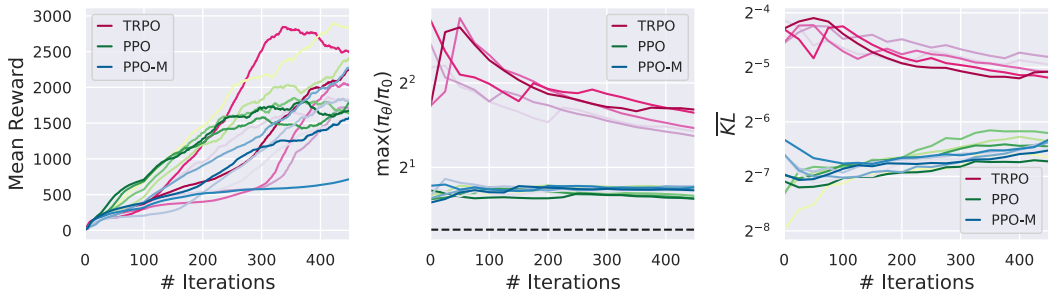
Figure 4: Per step mean reward, maximum ratio (c.f. (2)), mean KL, and maximum versus mean KL for agents trained to solve the MuJoCo Humanoid task. The quantities are measured over the state-action pairs collected in the *training step*. Each line represents a training curve from a separate agent. The 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 5; they are qualitatively nearly identical.



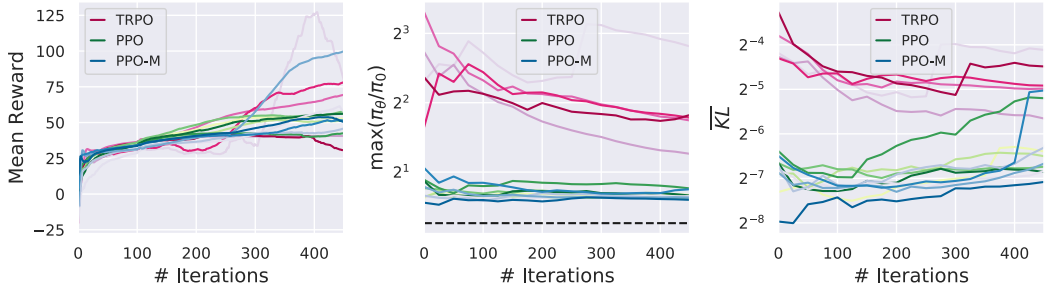
(a) Humanoid-v2 (heldout)



(b) Walker2d-v2 (heldout)

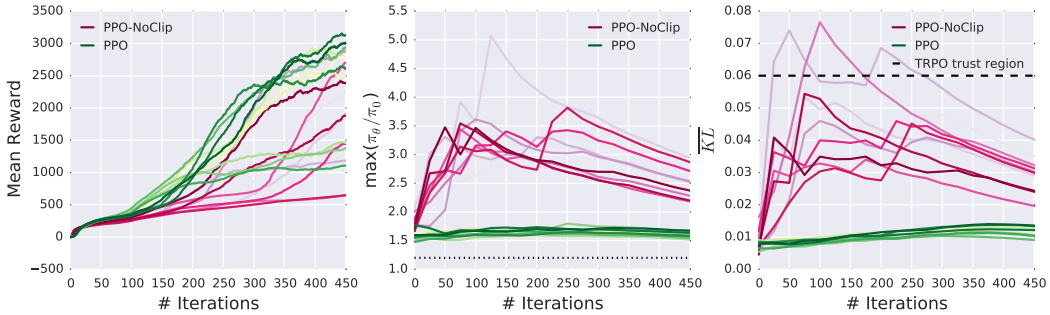


(c) Hopper-v2 (heldout)

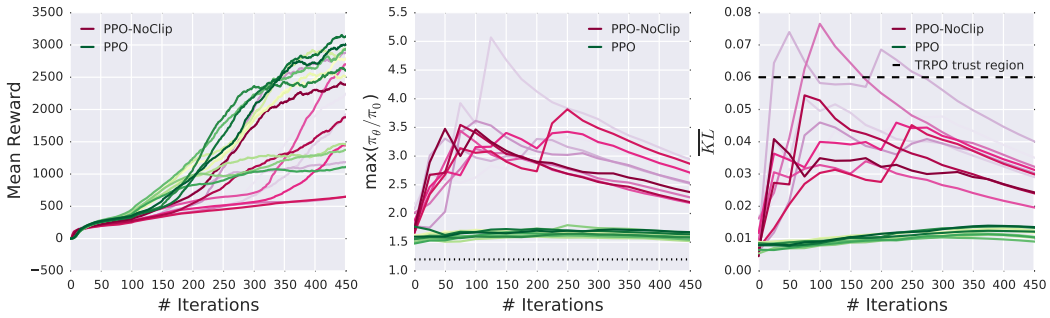


(d) Swimmer-v2 (heldout)

Figure 5: Per step mean reward, maximum ratio (c.f. (2)), 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).



(a) Walker2d-v2 (heldout)



(b) Hopper-v2 (heldout)

Figure 6: Per step mean reward, maximum ratio (c.f. (2)), mean KL, and mean KL for PPO and PPO-NoClip agents trained to solve the MuJoCo Swimmer-v2 (top) and Walker2d-v2 (bottom) task. The quantities are measured over the state-action pairs collected in *heldout steps*: i.e., these state-action pairs were sampled independently of those used to construct the steps. Each line represents a training curve from a separate agent. PPO-NoClip represents the PPO algorithm, with code-level optimizations, but without any clipping. From the plots we can see that the agents maintain a bounded trust region (indeed, more bounded than the trust region of the best TRPO agent, shown as the black line on the right), and achieve high reward, despite not using the PPO clipping step at all.