An empirical study of the convergence behavior of proximal policy optimization

Joshua Ivanhoe
Operations Research Centre
Massachusetts Institute of Technology
jivanhoe@mit.edu

Jordan Knight
Operations Research Centre
Massachusetts Institute of Technology
knightj@mit.edu

Cameron Hickert
John A. Paulson School of Engineering and Applied Sciences
Harvard University
cameron_hickert@hks.harvard.edu

Abstract

The Proximal Policy Optimization (PPO) algorithm is one of the most successful and stable variants of on-policy learning and is widely used by reinforcement learning practitioners today. It uses a gradient clipping function to limit the amount the policy gradient can be updated at one time, which helps prevent the agent from updating into an unrecoverable policy space. This is achieved by a clamp function governed by a parameter ϵ which controls the window around which the gradient will be clipped. In this paper we explore the empirical performance of alternative clipping functions as well as the behavior of the clipping parameter under noisy environments. We create two toy environments to facilitate principled experimentation and explore a large number of parameter and environment combinations. We find that the agent shows strong learning behavior across all three clipping functions under a variety of conditions. We find that scheduled annealing of the clipping parameter can achieve performance on-par with an optimal fixed ϵ without the need for tuning.

1 Introduction

1.1 Motivation

Policy gradient methods are one of the major classes of contemporary reinforcement learning (RL) algorithms. They provide several advantages when compared to off-policy methods, including the ability to optimize an explicitly stochastic policy, and more easily allow for continuous output spaces. One of the downsides of these methods, however, is that the policy gradients themselves can be highly unstable, especially in the face of noisy or sparse reward signals. These types of rewards are quite common in real-world RL tasks, and so it is important that a candidate method is able to perform well in these scenarios. Proximal policy optimization (PPO) is one of the most popular choices for on-policy RL in modern applications, due to its relatively stable learning behavior on a variety of tasks. Despite this, there remain several areas for inquiry into the behavior of PPO, including the optimal approach to tuning the clipping parameter ϵ, the performance of the algorithm under noisy or sparse conditions, and possible alternatives to the clamp clipping function. We aim to explore these three aspects of PPO in controlled experiments to gain a deeper understanding of where PPO performs well and where it might stand to be improved.

1.2 Overview

In this paper, we investigate the convergence behavior of PPO, including with novel modifications, via an in-depth empirical study. The structure of paper is as follows. Section 2 provides a review of PPO's current approach, before introducing two potential augmentations: (i) alternative clipping functions and (ii) annealed clipping. Section 3 details the experimentation we performed, in terms of environments, methodology and implementation details. Section 4 outlines the results of this experimentation, with a focus on the how the proposed augmentations affect convergence speed under various sparsity and noise regimes. Section 5 concludes the paper by providing a summary of our key findings and suggesting directions for future research. Supplementary materials can be found in the Appendix.

2 Approach

2.1 Preliminaries

We begin by providing an outline of vanilla proximal policy optimization methods. Consider a Markov decision process (MDP) $\langle \mathcal{S}, \mathcal{A}, f, r, \gamma \rangle$ with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition density $f : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and discount $\gamma \in (0, 1)$. We wish to learn a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to [0, 1]$, parameterized by θ, such that the objective

$$J(\theta) = \mathbb{E}_\theta \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \right] \tag{1}$$

is maximized. By the policy gradient theorem (Sutton et al., 1999; Marbach and Tsitsiklis, 2001) we have that

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta \left[A_\theta(s, a) \nabla_\theta \log \pi_\theta(a|s) \right], \tag{2}$$

where A_θ is the advantage function under policy π_θ. To improve the data-efficiency of on-policy methods, it is desirable to take multiple gradient steps per sample. When a sample from the old policy $\pi_{\theta'}$ is used, the policy gradient is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\theta'} \left[A_\theta \nabla_\theta \rho_{\theta'}(\theta) \right], \tag{3}$$

where we have defined the policy probability ratio function

$$\rho_{\theta'}(\theta) \triangleq \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} \tag{4}$$

and omitted the explicit states and actions for conciseness (see Appendix A.1 for further details). However, in practice, we only have access to advantage estimates under $\pi_{\theta'}$, which we denote as $\hat{A}_{\theta'}$. As a result, the objective estimator

$$\hat{J}_{\theta'}(\theta) = \hat{A}_{\theta'} \rho_{\theta'}(\theta) \tag{5}$$

is biased when $\theta \neq \theta'$. This motivates the approach of trust region policy optimization (TRPO) (Schulman et al., 2015), which considers the constrained optimization problem

$$\max \hat{J}_{\theta'}(\theta) \text{ s.t. } \mathcal{D}_{\text{KL}} \left(\pi_\theta || \pi_{\theta'} \right) \leq \delta, \tag{6}$$

where $\mathcal{D}_{\text{KL}}$ denotes the Kullback–Leibler divergence between the distributions. This can be solved approximately using the conjugate gradient method, which is a second-order algorithm. To obtain a problem that can be solved with a first-order algorithm, we can relax the constraint into the objective

$$\hat{J}^{\text{TRPO}}_{\theta'}(\theta) = \hat{A}_{\theta'} \rho_{\theta'}(\theta) - \beta \mathcal{D}_{\text{KL}} \left(\pi_\theta || \pi_{\theta'} \right), \tag{7}$$

where β is a hyperparameter. However, Schulman et al. (2017) observed that it is difficult to find a fixed value for β that performs consistently over the course of learning for a single task, let alone across multiple environments. They instead propose optimizing the surrogate objective

$$\hat{J}^{\text{PPO}}_{\theta'}(\theta) = \min \left\{ \hat{A}_{\theta'} \rho_{\theta'}(\theta), \hat{A}_{\theta'} \sigma \left(\rho_{\theta'}(\theta) \right) \right\}, \tag{8}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the clipping function. Note that the minimum of clipped and unclipped estimators is taken to ensure that the surrogate objective is a lower bound on the unclipped objective. Introducing the clipping parameter ϵ, Schulman et al. (2017) define the clipping function as

$$\sigma(\rho) = \begin{cases} 1-\epsilon, & \rho \leq 1-\epsilon, \\ 1+\epsilon, & \rho \geq 1+\epsilon, \\ \rho, & \text{otherwise.} \end{cases} \tag{9}$$

This clipping function is constructed to remove the incentive to update the policy probability ratio beyond the range $[1-\epsilon, 1+\epsilon]$, as the gradient estimator becomes zero outside this region. For clarity, we refer to (9) as the *clamp* clipping function from here on.

2.2 Clipping function variants

Given the PPO framework, a natural question that arises is whether the clamp is the best choice of clipping function. To address this, we must first define the necessary properties of a clipping function, in order to appreciate what viable alternatives may exist. Namely, in designing a clipping function σ parameterized by clipping parameter ϵ, we would like it have following properties:

(i) It should be bounded, such that $|\sigma(\rho) - 1| \leq \epsilon,\ \forall \rho \geq 0$;

(ii) It should be conservative, in the sense that $|\sigma(\rho) - 1| \leq |\rho - 1|\,,\ \forall \rho \geq 0$;

(iii) It should preserve the consistency of the objective and gradient estimators at $\theta = \theta'$, by ensuring that $\sigma(1) = \sigma'(1) = 1$; and

(iv) It should have a bounded and nonempty subgradient $\forall \rho \geq 0$.

Properties (i) and (ii) capture the basic goal of the clipping function to prevent large changes in the policy probability ratio. Property (iii) ensures that the first update is unaffected by the clipping function. Note also that , in property (iii), we denote by σ' the derivative of σ, which is assumed to be defined at $\rho = 1$. Property (iv) is required for the clipping function to be compatible with subgradient descent and its various implementations using automatic differentiation software (*e.g.* `PyTorch`).

We remark that the clamp is the simplest choice of clipping function that fulfils all four of these properties. However, as highlighted by Wang et al. (2019), one shortcoming of the clamp is that it still may result in gradient updates that move policy probability ratio outside the range $[1-\epsilon, 1+\epsilon]$. Moreover, the clamp does not provide a corrective mechanism to move the ratio back within this desired range should such an 'overzealous' update occur, though it will not perturb the ratio further. This is because, under mild assumptions, the magnitude of the change to the policy probability ratio when the clipped estimator is used is of the order $\mathcal{O}\left(\sigma'(\rho)\right)$ (see Appendix A.2 for further details). To overcome this problem, Wang et al. (2019) propose a *rollback* clipping function of the form

$$\sigma(\rho) = \begin{cases} (1-\epsilon)(1-\alpha) + \alpha\rho, & \rho \leq 1-\epsilon, \\ (1+\epsilon)(1+\alpha) - \alpha\rho, & \rho \geq 1+\epsilon, \\ \rho, & \text{otherwise,} \end{cases} \tag{10}$$

where α is an additional hyperparameter that controls the aggressiveness of rollback behavior. The intuition guiding this parameterization is that outside of the range $[1-\epsilon, 1+\epsilon]$, the derivative of the rollback clipping function becomes

$$\sigma'(\rho) = \alpha \text{sign}(1-\rho), \tag{11}$$

meaning the ratio is corrected towards the desired range. Consequently, for suitable choices of α, the policy probability ratio under the rollback scheme remains strictly closer to one, relative to the vanilla PPO method with the clamp clipping function. We also propose the following novel clipping function:

$$\sigma(\rho) = \epsilon \tanh\left(\frac{\rho - 1}{\epsilon}\right) + 1, \tag{12}$$

which we refer to as the *tanh* clipping function. This function satisfies the necessary properties outlined above, while also smoothly tapering the clipped ratio and adding preemptive tapering within the range $[1-\epsilon, 1+\epsilon]$. Accordingly, changes to the policy probability ratio become smaller, relative to

those resulting from the clamp clipping function, as the boundary of the desired range is approached. Figure 1 illustrates the differences between the three clipping function variants discussed.

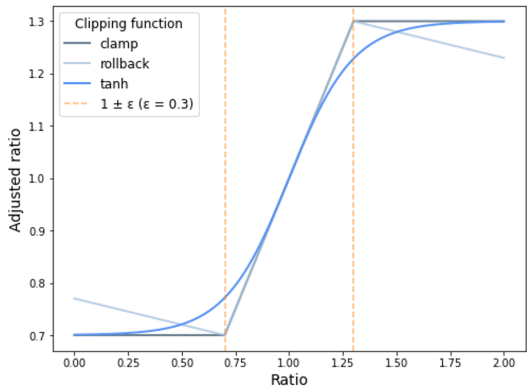

Figure 1: Visualization of clipping function variants (with clipping parameter $\epsilon = 0.3$)

2.3 Annealed clipping

A practical consideration is how the choice of clipping parameter affects the performance of PPO. Ideally, we would like to have a single value for the clipping parameter that leads to stable learning across a variety of tasks. To this end, Schulman et al. (2017) recommend $\epsilon = 0.2$, based on a coarse grid search for seven simulated robotics tasks implemented in the OpenAI Gym (Brockman et al., 2016). However, this study is far from comprehensive and they do no provide data on the variance of the performance between tasks. An alternative approach is to anneal the clipping parameter over some predetermined range $[\epsilon^{\min}, \epsilon^{\max}]$, rather than tuning a single, problem-specific value. This technique is often applied to other hyperparameters in RL algorithms, such as the learning rate and the exploration parameter ϵ (unrelated to the clipping parameter ϵ) in ϵ-greedy algorithms. In a simple linear annealing schedule, the clipping parameter at iteration k is given by

$$\epsilon_k = \epsilon^{\min} + (\epsilon^{\max} - \epsilon^{\min}) \left(\frac{\Pi - (k \mod \Pi)}{\Pi} \right), \tag{13}$$

where Π is the period of the schedule. A more sophisticated annealing scheme may instead follow a sinusoidal schedule, with the clipping parameter at iteration k is given by

$$\epsilon_k = \epsilon^{\min} + \frac{1}{2}(\epsilon^{\max} - \epsilon^{\min}) \left(1 + \cos \left(\frac{\pi(k \mod \Pi)}{\Pi} \right) \right). \tag{14}$$

This has the effect of concentrating the annealed parameter near the boundary values more frequently. The bounds themselves can also be tightened over time, via the updates

$$\epsilon^{\max}_{k+1} = \omega^{\max} \epsilon^{\max}_k + (1 - \omega^{\max}) \epsilon^{\min}_k, \tag{15}$$

$$\epsilon^{\min}_{k+1} = \omega^{\min} \epsilon^{\min}_k + (1 - \omega^{\min}) \epsilon^{\max}_{k+1}, \tag{16}$$

where $\omega^{\max}, \omega^{\min} \in (0, 1]$ are respectively the decay and growth rates of upper and lower bounds respectively. In particular, note that the setting $\omega^{\max} < 1$ and $\omega^{\min} = 1$ corresponds to a typical cooling schedule. Figure 2 provides a comparative illustration of the behavior of these annealing schemes.

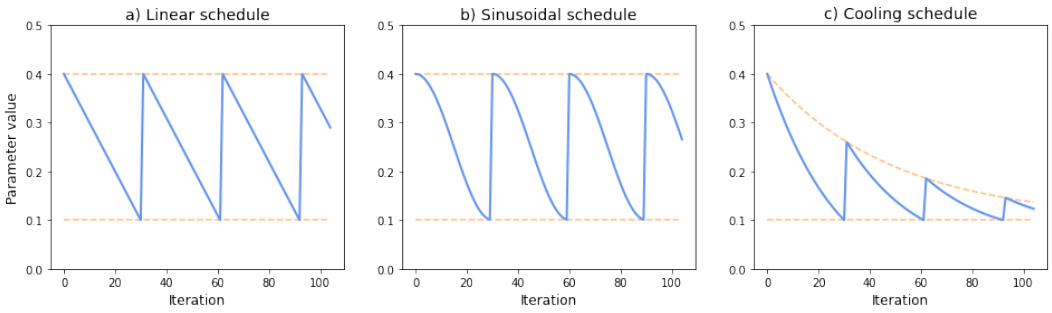

Figure 2: Visualization of example annealing schedules

In addition to the computational savings afforded by avoiding a grid search to tune the clipping parameter, there is also an intuitive motivation for employing annealing. Clipping mitigates the ability of 'overzealous' updates to destabilize the learning process. However, if the clipping parameter is too tight, it hinders necessary learning from happening. Already, PPO itself is a balance between these two extremes: unstable learning on the one hand, and impractically slow learning on the other. With this in mind, annealing the clipping parameter is an alternative attempt to balance these competing interests. At its best, the approach would allow the learner to make larger updates to find promising regions, then enforce tighter clipping to prevent destabilizing updates. The periodic pattern could allow this process to repeat, while the narrowing outer bounds gradually push the learner further towards stability. Nonetheless, a potential downside of this approach is that the annealing schedule itself introduces many hyperparameters that are themselves impractical to tune and must be set using intuition.

3 Experimentation

3.1 Environments

To enable principled and efficient experimentation, we designed two toy learning environments, which we refer to as `GoalFinder` and `PhysicsLander`. Both of these environments consist of n-dimensional continuous worlds in which the agent must move from the origin $\mathbf{0}_n$ to the goal $g = \mathbf{1}_n$. In `GoalFinder`, the agent has direct control over its position and hence its only task is *find the goal*. The `PhysicsLander` environment poses a more challening task by only giving the agent control over its acceleration, with the dynamics governed by discretized Newtonian physics, meaning it must learn how to *land on the goal*. The attributes of each environment are summarized in Table 1. Note that a position dependent force field F is also be introduced to the `PhysicsLander`. In experimentation, this was set to a constant force akin to a local approximation of gravity.

In both environments, the reward is based on the distance from the current position to the goal. The parameter λ determines the sparsity of the reward, with raw sparse reward recovered as $\lambda \to \infty$. The $\frac{1}{\sqrt{n}}$ factor is included to decouple the reward sparsity from the dimensionality of the environment. The second term in the reward function represents additive noise, with the parameter σ controlling the magnitude of the noise and the random variable z_t sampled from a distribution with zero mean. In our experimentation, we consider two sampling distributions for the reward noise: (i) normally distributed noise, where $z_t \overset{\text{i.i.d.}}{\sim} \text{Normal}(0, 1)$; and (ii) adversarial noise, where $z_t \overset{\text{i.i.d.}}{\sim} \text{Uniform}(\{-1, 1\})$. The adversarial noise is so named because, given a finite support, the perturbation always takes its maximum magnitude. Altogether, this reward function allows us to easily parameterize the difficulty of the environment in terms of the sparsity parameter λ and the noise parameter σ, in conjunction with the dimensionality of the environment.

Table 1: Summary of `GoalFinder` and `PhysicsLander` environment attributes

Attribute	`GoalFinder`	`PhysicsLander`
State	Position: $s_t \in \mathbb{R}^n$	Position/velocity: $s_t = (x_t, v_t) \in \mathbb{R}^{2n}$
Action	Displacement: $a_t \in \mathbb{R}^n$	Acceleration: $a_t \in \mathbb{R}^n$
Transition function	$s_{t+1} = s_t + a_t$	$x_{t+1} = x_t + v_t,\ v_{t+1} = v_t + a_t + F(x_t)$
Reward	$\exp\left(-\frac{\lambda}{\sqrt{n}} \left\|s_{t+1} - g\right\|_2^2\right) + \sigma z_t$	$\exp\left(-\frac{\lambda}{\sqrt{n}} \left\|x_{t+1} - g\right\|_2^2\right) + \sigma z t$

3.2 Methodology

Our experiments consisted of a comparative grid search over the parameters outlined in Table 2 for `GoalFinder` and Table 3 for `PhysicsLander`. These grids were selected based on coarse preliminary experimentation. Note that, for simplicity, we only considered a linear annealing schedule with with a period of 20 iterations for the annealed clipping experiments. For each combination of parameters, we performed five trials across different seeds. We evaluated the performance of the algorithm in terms of its convergence speed. Given that the reward for our environments is bounded, ignoring noise, we defined the number of *iterations to converge* as the minimum number of iterations required to obtain a mean batch reward above 80% of the maximum reward. It should be noted that this is a rather arbitrary definition, based on empirical observation of learning curves. Future work could consider more general notions of convergence.

Overall, we investigated 1008 parameter combinations for the `GoalFinder` environment, and 324 for the `PhysicsLander` environment. With 5 trials per parameter combination, 150 iterations per trial and 64 trajectories per iteration, this amounts to 48.4 million generated trajectories for `GoalFinder` and 15.6 million generated trajectories for `PhysicsLander`. The disparity results from the fact that our grid search on the `GoalFinder` environment informed our grid selection for the `PhysicsLander` experiments, enabling us to avoid more parameter combinations uninteresting to our analysis.

Table 2: `GoalFinder` experiments

Environment Parameters	
Number of dimensions	2, 3, 4
Sparsity parameter	2
Reward noise parameter	0.0, 0.25, 0.5, 1.0
Noise distribution	normal, adversarial
PPO Parameters	
Clipping parameter	0.1, 0.2, 0.3, 0.4, 0.1-0.4 (annealed), 0.2-0.3 (annealed)
Clipping function	clamp, rollback, tanh

Table 3: `PhysicsLander` experiments

Environment Parameters	
Sparsity parameter	2, 4, 8
Reward noise parameter	0.0, 0.25, 0.5
Noise distribution	normal, adversarial
PPO Parameters	
Clipping parameter	0.2, 0.4, 0.1-0.4 (annealed)
Clipping function	clamp, rollback, tanh

3.3 Implementation details

We ran our experiments on the AWS Elastic Cloud Compute infrastructure, using m4.4xlarge EC2 instances configured with the Deep Learning AMI. Our implementation of PPO[1] utilized `PyTorch` with CUDA and parallelized trajectory generation to accelerate the training process. For the actor and critic models, we used a fully-connected neural network architecture with two hidden layers of sizes 64 and 32 respectively, and ReLU activation functions. This architecture was kept fixed throughout all experimentation. Batch gradient descent was performed using the Adam optimizer (Kingma and Ba, 2014), with a learning rate of 3×10^{-4} and 5 epochs per batch.

4 Results

4.1 Empirical performance of clipping function variants

The empirical performance of the clamp, rollback and tanh clipping functions on the `GoalFinder` environment is summarized in Figure 4. Note that the least challenging environment configuration is in the top left-hand corner and the most challenging is in the bottom right-hand corner. For each clipping function and environment configuration combination, performance is measured for the best corresponding clipping parameter setting. While these results suggest that the tanh and rollback clipping functions may offer some benefit over the standard clamp clipping function in high-dimensional, low-noise settings, there appears to be little difference in the overall performance of the three variants. Analogous results for the `PhysicsLander` environment are shown in Figure 7 in the Appendix and support the same conclusion. Moreover, the training curves in Figure 3 demonstrate that the clipping function variants tend to exhibit qualitatively similar learning behaviour.

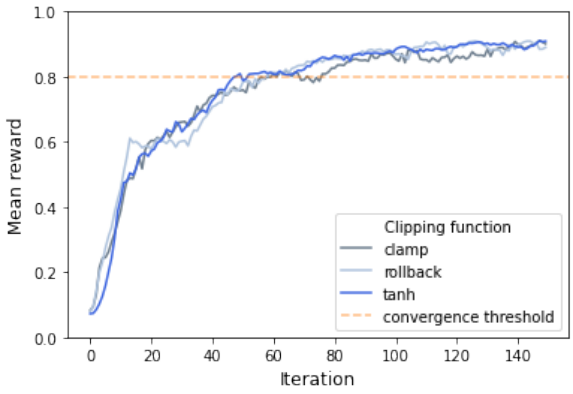

Figure 3: Example learning curves for clipping function variants on `GoalFinder` with $n = 2$ and $\sigma = 0$ (based on median performance over 5 trials)

[1]Source code can be accessed via the public GitHub repository: `https://github.com/jivanhoe/proximal-policy-optimization`. For reproducibility, we have included an environment.yml file that contains the necessary packages and versions to emulate the Conda environment used for the experiments.

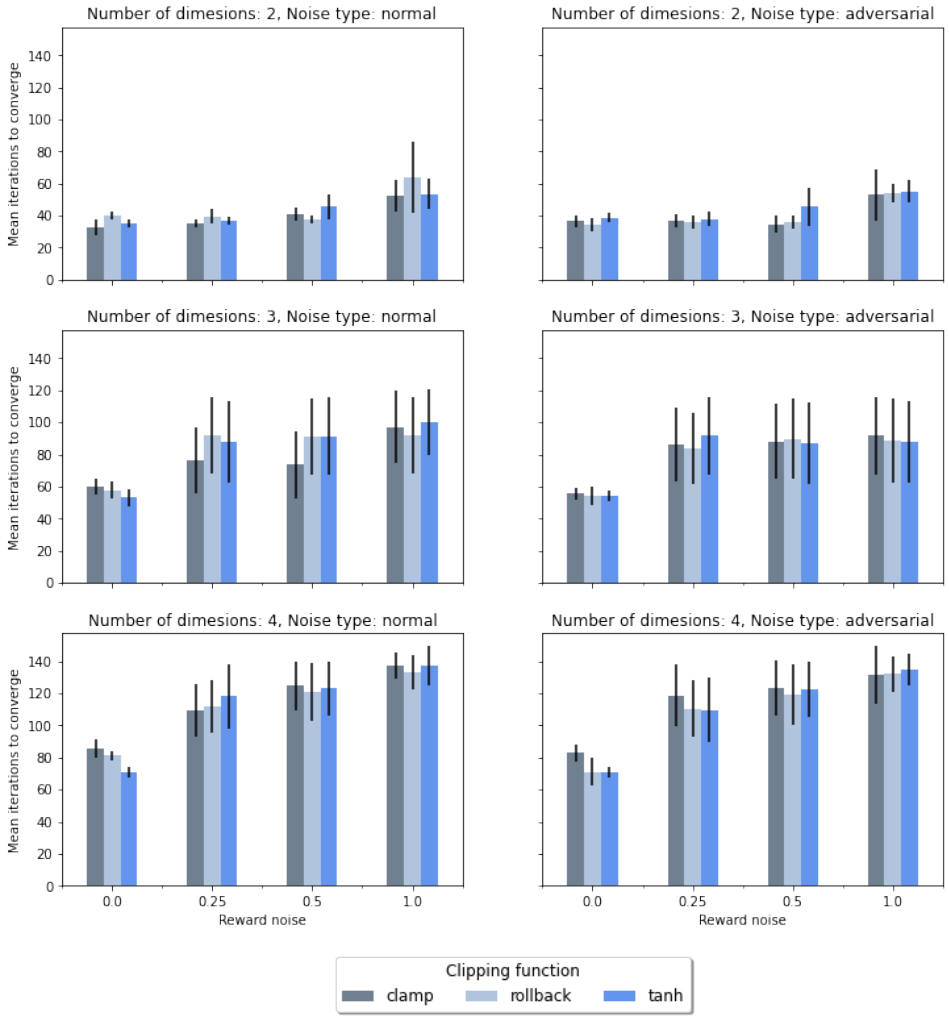

Figure 4: The impact of clipping function on convergence speed for various settings of the `GoalFinder` environment (error bars represent 95% confidence intervals based on 5 random trials)

4.2 The impact of clipping parameter on convergence speed

The impact of the clipping parameter on convergence speed for the `GoalFinder` environment is illustrated in Figure 5. As above, the least challenging environment configuration is in the top left-hand corner and the most challenging is in the bottom right-hand corner, with the performance for each clipping parameter and environment configuration pairing measured using the best corresponding clipping function. We observe that the clipping parameter setting has a significant effect on the convergence speed of the algorithm. In this case, it was observed that a larger clipping parameter of 0.4 was generally the best value. Interestingly, this appears to be largely independent of the the magnitude or type of reward noise, or the dimensionality of the environment. This finding is consistent with the observation of Henderson et al. (2017) that larger clipping parameters tend to be more suitable for simpler architectures. Once again, analogous results for the `PhysicsLander` environment are shown in Figure 8 in the Appendix and reinforce this conclusion. Another important insight is that the annealed clipping schedules performed almost as well as the best fixed value. Furthermore, Figure 6 demonstrates that annealing the clipping parameter over a large range can produce learning behaviour that is qualitatively very similar to that achieved with best fixed value. This suggests that annealing can give near-optimal results without the need for a grid search to tune the clipping parameter.

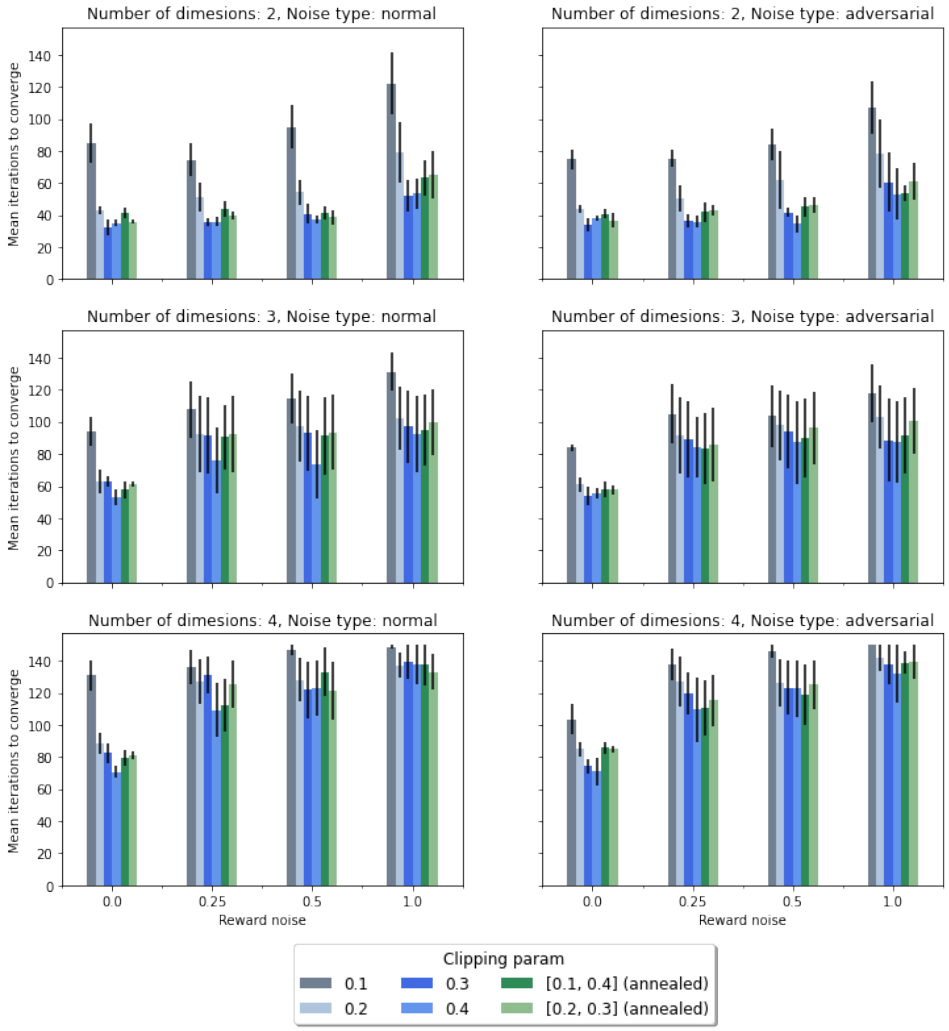

Figure 5: Impact of clipping parameter value/schedule on convergence speed for various settings of the `GoalFinder` environment (error bars represent 95% confidence intervals using on 5 random trials)

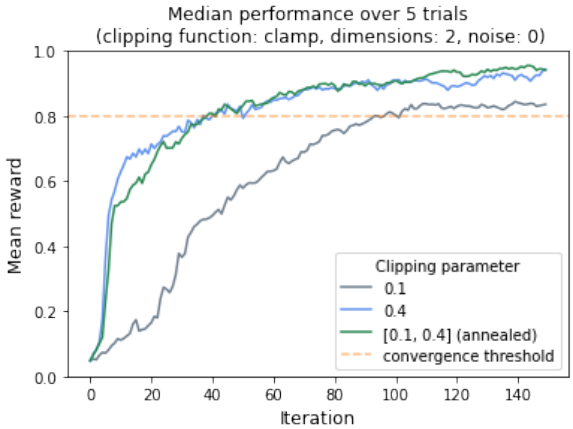

Figure 6: Example learning curves for different clipping parameter settings on `GoalFinder` with $n = 2$ and $\sigma = 0$ (based on median performance over 5 trials)

5 Conclusion

5.1 Key findings

First, we observe that the all three clipping function variants lead to similar learning behavior and convergence properties. The rollback and tanh functions may have some benefits when applied to low-noise, higher dimensional environments, but this would require additional tests on more complex learning problems to confirm. Second, we observe that annealing the clipping parameter ϵ according to a periodic schedule can be a viable alternative to tuning the ϵ parameter directly. We find that the annealed clipping schedules performed almost as well as the best fixed value in our environments, and do not require any hyperparameter tuning as long as the optimal clipping value exists somewhere in the range of the annealed parameter's schedule. This suggests that annealing the clipping parameter has the potential to become a simpler alternative to finding the right clipping parameter when training PPO. We did not find that noise had a significant impact on the learning problem and as a result all of the different clipping function and parameter combinations were able to learn fairly successfully even in a high-noise environment, although the time to convergence was generally increased. This is likely due to the fact that the problems were relatively easy learning problems in general, and so the addition of noise was not sufficient to make them unstable.

5.2 Future work

Given this initial exploration, a few areas of continued investigation are immediately interesting to us. There are several aspects of the PPO algorithm which we did not vary in our experiments. One significant part of PPO that has an impact on learning is the network architecture used, and we would like to try out several architecture sizes to determine how they interact with the different clipping functions and parameters. In addition, we use a simple method of calculating the discounted returns, but it would be interesting to test out Generalized Advantage Estimates on noisy environments to see how much improvement occurs.

We also found that our environments allowed for learning problems which were not sufficiently challenging as to provide a meaningful difference in the behavior of most of the parameter combinations, even with noisy rewards. It is therefore important that we replicate our analysis on more challenging benchmarks (such as Atari or OpenAI Gym environments), including those with discrete action spaces. We would also like to look into different ways to add noise to an environment, like non-deterministic transition functions.

Acknowledgments

We would like to thank Pulkit Agrawal and Anurag Ajay for their guidance throughout the course, without which this project would not be possible. We also would like to thank our colleagues for their consistent enthusiasm and many interesting in-class discussions.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *CoRR*, abs/1606.01540.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep reinforcement learning that matters. *CoRR*, abs/1709.06560.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

P. Marbach and J. N. Tsitsiklis. 2001. Simulation-based optimization of markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust region policy optimization. *CoRR*, abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA. MIT Press.

Yuhui Wang, Hao He, and Xiaoyang Tan. 2019. Truly proximal policy optimization. *CoRR*, abs/1903.07940.

A Technical asides

A.1 Note on the revised form of the policy gradient

We begin with the policy gradient in its canonical form (2). By importance sampling, we have that

$$\mathbb{E}_\theta \left[A_\theta(s,a) \nabla_\theta \log \pi_\theta(a|s) \right] = \mathbb{E}_{\theta'} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} A_\theta(s,a) \nabla_\theta \log \pi_\theta(a|s) \right]. \tag{17}$$

Noting that

$$\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} \nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta'}(a|s)} = \nabla_\theta \rho_{\theta'}(\theta), \tag{18}$$

we obtain the revised form of the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta \left[A_\theta \nabla_\theta \rho_{\theta'}(\theta) \right]. \tag{19}$$

A.2 Note on the order of the policy probability ratio change

We define the function

$$\phi_{\theta'}(\theta, \eta) \triangleq \rho_{\theta'} \left(\theta + \eta \nabla_\theta \hat{J}^{\text{PPO}}_{\theta'}(\theta) \right), \tag{20}$$

to represent the value of $\rho_{\theta'}$ after a gradient update from incumbent value θ with learning rate η. By the chain rule, we have that:

$$\begin{aligned} \partial_\eta \phi_{\theta'}(\theta, 0) &= \left\langle \nabla_\theta \rho_{\theta'}(\theta), \nabla_\theta \hat{J}^{\text{PPO}}_{\theta'}(\theta) \right\rangle, \\ &= \hat{A}_{\theta'} \sigma' \left(\rho_{\theta'}(\theta) \right) \left\| \nabla_\theta \rho_{\theta'}(\theta) \right\|_2^2. \end{aligned} \tag{21}$$

Given that the policy probability ratio change can be expressed as

$$\phi_\theta(\theta, \eta) - \phi_\theta(\theta, 0) = \eta \partial_\eta \phi_{\theta'}(\theta, 0) + \mathcal{O}(\eta^2), \tag{22}$$

if $1/\eta \sim \left\| \nabla_\theta \rho_{\theta'}(\theta) \right\|_2^2$, the change in the ratio is $\mathcal{O}\left(\sigma'(\rho_{\theta'}(\theta)) \right)$.

B Additional results

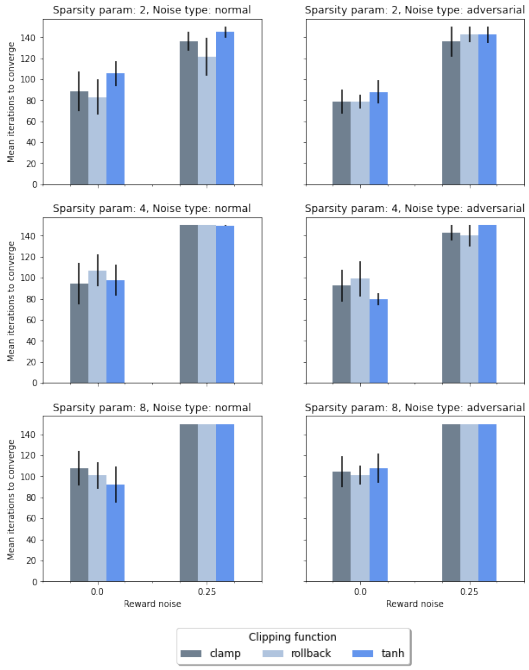

Figure 7: The impact of clipping function on convergence speed for various settings of the `PhysicsLander` environment (error bars represent 95% confidence intervals based on 5 random trials)

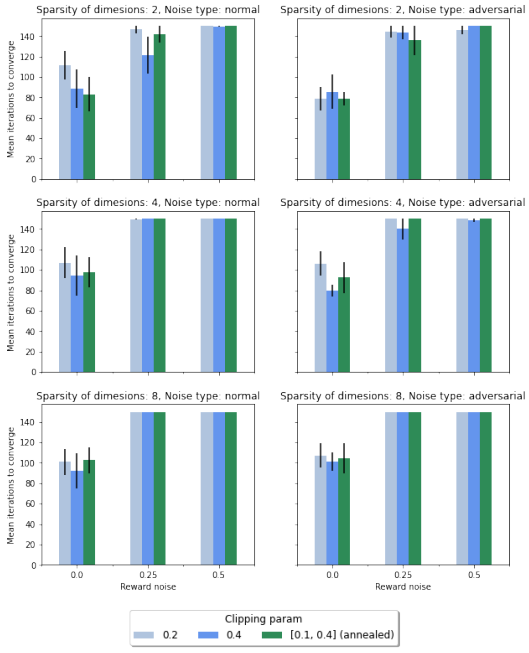

Figure 8: The impact of clipping parameter value/schedule on convergence speed for various settings of the `PhysicsLander` environment (error bars represent 95% confidence intervals based on 5 random trials)