PPO-CMA: PROXIMAL POLICY OPTIMIZATION WITH COVARIANCE MATRIX ADAPTATION

Anonymous authors
Paper under double-blind review

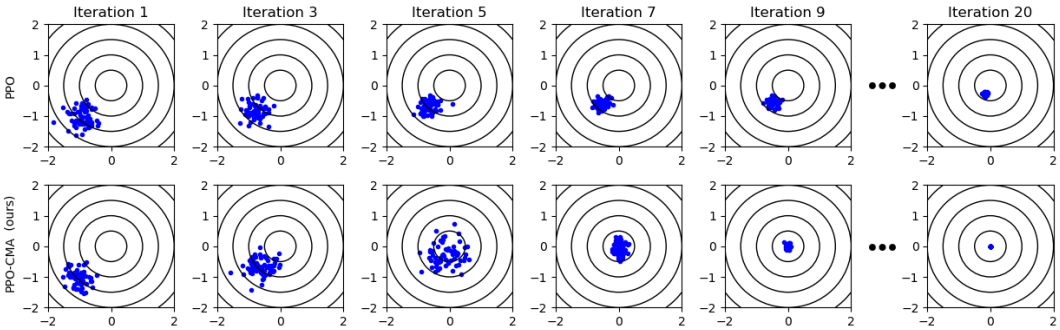

Figure 1: Comparison of Proximal Policy Optimization (PPO) and our proposed PPO-CMA method. Sampled actions $\mathbf{a} \in \mathbb{R}^2$ are shown in blue. In this "stateless" didactic example, the simple quadratic objective has the global optimum at the origin (details in Section 3.1). PPO shrinks the exploration variance prematurely, which leads to slow convergence. PPO-CMA dynamically expands the variance to speed up progress, and only shrinks the variance when close to the optimum.

ABSTRACT

Proximal Policy Optimization (PPO) is a highly popular model-free reinforcement learning (RL) approach. However, in continuous state and actions spaces and a Gaussian policy – common in computer animation and robotics – PPO is prone to getting stuck in local optima. In this paper, we observe a tendency of PPO to prematurely shrink the exploration variance, which naturally leads to slow progress. Motivated by this, we borrow ideas from CMA-ES, a black-box optimization method designed for intelligent adaptive Gaussian exploration, to derive PPO-CMA, a novel proximal policy optimization approach that can expand the exploration variance on objective function slopes and shrink the variance when close to the optimum. This is implemented by using separate neural networks for policy mean and variance and training the mean and variance in separate passes. Our experiments demonstrate a clear improvement over vanilla PPO in many difficult OpenAI Gym MuJoCo tasks.

1 INTRODUCTION

This paper proposes a new solution to the problem of policy optimization with high-dimensional continuous state and action spaces. This is a problem that has a long history in computer animation, robotics, and machine learning research. More specifically, our work falls in the domain of simulation-based Monte Carlo approaches; instead of operating with closed-form expressions of the control dynamics, we use a black-box dynamics simulator, sample actions from some distribution, simulate the results, and then adapt the sampling distribution. In recent years, such approaches have achieved remarkable success in previously intractable tasks such as real-time locomotion control of (simplified) biomechanical models of the human body (Wang et al. (2010); Geijtenbeek et al. (2013); Hämäläinen et al. (2014; 2015); Liu et al. (2016); Duan et al. (2016); Rajamäki & Hämäläinen (2017)). In 2017, Proximal Policy Optimization (PPO) provided the first demonstration of a neural

network policy that enables a simulated humanoid not only to run but also to rapidly switch direction and get up after falling (Schulman et al. (2017)). Previously, such feats had only been achieved with more computationally heavy approaches that used simulation and sampling not only during training but also in run-time (Hämäläinen et al. (2014; 2015); Rajamäki & Hämäläinen (2017)).

PPO has been quickly adopted as the default RL algorithm in popular frameworks like Unity Machine Learning Agents [1] and TensorFlow Agents (Hafner et al. (2017)). It has also been extended and applied to even more complex humanoid movement skills such as kung-fu kicks and backflips (Peng et al. (2018)). Outside the continuous control domain, it has demonstrated outstanding performance in complex multi-agent video games [2]. However, like many other reinforcement learning methods, PPO can be sensitive to hyperparameter choices and difficult to tune (Henderson et al. (2017)). In experiments by ourselves and our colleagues, we have also noticed a tendency to get stuck in local optima.

In this paper, we make the following contributions:

- We provide visualizations and evidence of how PPO's exploration variance can shrink prematurely, which leads to slow progress or getting stuck in local optima. Figure 1 illustrates this in a simple didactic example. Subsequently, we discuss how a similar exploration problem is solved in the black-box optimization domain by Covariance Matrix Adaptation Evolution Strategy (CMA-ES). CMA-ES dynamically expands the exploration variance on objective function slopes and only shrinks the variance when close to the optimum.
- We show how exploration behavior similar to CMA-ES can be achieved in RL with simple changes, resulting in our PPO-CMA algorithm visualized in Figure 1. As elaborated in Section 4, a key idea of PPO-CMA is to use separate neural networks for policy mean and variance, and train the mean and variance in separate passes. Our experiments show a significant improvement over PPO in many OpenAI Gym MuJoCo tasks (Brockman et al. (2016)).

In Appendix A, we also investigate solving the variance adaptation problem with simple variance clipping and entropy regularization. The entropy regularization was suggested by Schulman et al. (2017) but not analyzed in detail. Our results suggest that both approaches can help but they are sensitive to tuning parameters.

2 Preliminaries

2.1 Reinforcement Learning

Algorithm 1 Episodic On-policy Reinforcement Learning (high-level summary)

```
for iteration=1,2,... do
    while iteration simulation budget N not exceeded do
        Reset the simulation to a (random) initial state
        Run agent on policy π_θ for T timesteps or until a terminal state
    end while
    Update policy parameters θ based on the observed experience [s_i, a_i, r_i, s'_i]
end for
```

We consider the discounted formulation of the policy optimization problem, following the notation of Schulman et al. (2015b). At time t, the agent observes a state vector $\mathbf{s}_t$ and takes an action $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, where π_θ denotes a stochastic policy parameterized by θ, e.g., neural network weights. This results in observing a new state $\mathbf{s}'_t$ and receiving a scalar reward r_t. The goal is to find θ that maximizes the expected future-discounted sum of rewards $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where γ is a discount factor in the range $(0, 1)$. A lower γ makes the learning prefer instant gratification instead of long-term gains.

[1] https://github.com/Unity-Technologies/ml-agents

[2] https://blog.openai.com/openai-five/

The original PPO and the PPO-CMA proposed in this paper collect experience tuples $[\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i]$ by simulating a number of *episodes* in each optimization iteration. For each episode, an initial state $\mathbf{s_0}$ is sampled from some application-dependent stationary distribution, and the simulation is continued until a terminal (absorbing) state or a predefined maximum episode length T is reached. After an iteration simulation budget N is exhausted, θ is updated. This is summarized in Algorithm 1.

2.2 Policy gradient with advantage estimation

Policy gradient methods update policy parameters by estimating the gradient $\mathbf{g} = \nabla_\theta \mathbb{E}[\sum_t^\infty \gamma^t r_t]$. There are different formulations of the gradient, of which PPO uses the following:

$$\mathbf{g} = \mathbb{E}\Big[\sum_{t=0}^{\infty} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\Big], \tag{1}$$

where A^π is the so-called *advantage function*.

Intuitively, the advantage function is positive if an explored action yields better rewards than expected. Updating θ in the direction of $\mathbf{g}$ makes negative advantage actions less likely and positive advantage actions more likely (Schulman et al. (2015b)). In practice, using modern compute graph frameworks like TensorFlow (Abadi et al. (2016)), one often does not directly operate on the gradients, but instead uses an optimizer like Adam (Kingma & Ba (2014)) to minimize the corresponding loss

$$\mathcal{L} = -\frac{1}{M} \sum_{i=1}^{M} A^\pi(\mathbf{s}_i, \mathbf{a}_i) \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i), \tag{2}$$

where i denotes minibatch sample index and M is minibatch size. In summary, this type of policy gradient RL simply requires a differentiable expression of π_θ and a way to measure A^π for each explored state-action pair.

More specifically, the advantage function is defined as:

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t). \tag{3}$$

Here, V^π is the state value function, i.e., the expected future-discounted sum of rewards for running the agent on-policy starting from state $\mathbf{s}_t$. $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ is the state-action value function, i.e., expected sum of rewards for taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and then following the policy, $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^\pi(\mathbf{s}_{t+1})$. Thus, the advantage can also be expressed as

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t). \tag{4}$$

In practice, V^π is usually approximated by a critic network trained with the observed rewards summed over simulation trajectories. However, plugging such an approximation directly to Equation 4 tends to be unstable due to approximation bias. Instead, same as PPO, we use Generalized Advantage Estimation (GAE) (Schulman et al. (2015b)), which is a simple but effective way to estimate A^π such that one can trade variance for bias.

3 Understanding variance adaptation in Gaussian policy optimization

This paper focuses on the case of continuous control using a Gaussian policy. In other words, the policy network outputs state-dependent mean $\mu_\theta(\mathbf{s})$ and covariance $\mathbf{C}_\theta(\mathbf{s})$ for sampling the actions. The covariance defines the exploration-exploitation balance. In practice, one often uses a diagonal covariance matrix parameterized by a vector $\mathbf{c}_\theta(\mathbf{s}) = diag(\mathbf{C}_\theta(\mathbf{s}))$. In the most simple case of isotropic unit Gaussian exploration, $\mathbf{C} = \mathbf{I}$, the loss function in Equation 2 becomes:

$$\mathcal{L} = -\frac{1}{M}\sum_{i=1}^{M} A^{\pi}(\mathbf{s}_i, \mathbf{a}_i)||\mathbf{a}_i - \mu_\theta(\mathbf{s})||^2, \tag{5}$$

Following the original PPO paper, this paper uses diagonal covariance. This results in a slightly more complex loss function:

$$\mathcal{L} = -\frac{1}{M}\sum_{i=1}^{M} A^{\pi}(\mathbf{s}_i, \mathbf{a}_i)\sum_j[(a_{i,j} - \mu_{j;\theta}(\mathbf{s}_i))^2/c_{j;\theta}(\mathbf{s}_i) + 0.5\log c_{j;\theta}(\mathbf{s}_i)], \tag{6}$$

where i indexes over a minibatch and j indexes over action variables.

3.1 THE INSTABILITY CAUSED BY NEGATIVE ADVANTAGES

To gain an intuitive understanding of the training dynamics, it is important to note the following:

- Equation 5 indicates that the *policy is trained to approximate the sampled actions*, with each action weighted by A^{π}. Equation 5 uses an $L2$ loss, while Equation 6 uses a Bayesian loss that allows the network to model aleatoric uncertainty, expressed as the free covariance parameters (Kendall & Gal (2017)).
- As elaborated below, *actions with a negative A^{π} may cause instability*, especially when one considers training for several epochs at each iteration using the same data; as pointed out by Schulman et al. (2017), this would be desirable to get the most out of the collected experience.

For actions with positive advantages, training produces a Gaussian approximation where more weight is given to actions with higher advantages. In other words, the policy Gaussian will not diverge outside the proximity of the sampled actions. For negative advantages, *each gradient update drives the policy Gaussian further away from the sampled actions*. This can easily result in divergence, as shown at the top of Figure 2. In contrast, the bottom of Figure 2 shows how the updates are stable if using only the positive advantage actions. However, in this case the exploration variance shrinks prematurely, leading to poor final convergence.

Similar to Figure 1, Figure 2 depicts a "stateless" didactic example, a special case of Algorithm 1 that allows clear visualization of how the action distribution adapts. We set $\gamma = 0$, which simplifies the policy optimization objective $\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r_t] = \mathbb{E}[r_0] = \mathbb{E}[r(\mathbf{s}, \mathbf{a})]$, where $\mathbf{s}, \mathbf{a}$ denote the first state and action of an episode. Further, we use a state-agnostic $r(\mathbf{a}) = -\mathbf{a}^T\mathbf{a}$. Thus, $Q^{\pi}(\mathbf{s}, \mathbf{a}) = -\mathbf{a}^T\mathbf{a}$, $V^{\pi}(\mathbf{s}) = V^{\pi} = \mathbb{E}[r(\mathbf{a})] \approx \frac{1}{N}\sum_{i=0}^{N-1} r(\mathbf{a}_i)$, where i is episode index, and we can compute A^{π} directly using Equation 3. As everything is agnostic of agent state, a simulator is not even needed and one can feed an arbitrary constant input to a policy network. Equivalently, one can simply replace the policy network with optimized variables for the mean and variance of the action distribution.

3.2 PROXIMAL POLICY OPTIMIZATION: VARIANCE ADAPTATION PROBLEM

The basic idea of PPO is that one performs minibatch policy gradient updates for several epochs on the data from each iteration of Algorithm 1, while limiting changes to the policy such that it stays in the proximity of the sampled actions (Schulman et al. (2017)). PPO is a simplification of Trust Region Policy Optimization (TRPO) (Schulman et al. (2015a)), which uses a more computationally expensive approach to achieve the same.

The original PPO paper proposes two variants: 1) using a loss function that penalizes KL-divergence between the old and updated policies, and 2) using the so-called clipped surrogate loss function that limits the likelihood ratio of old and updated policies $\pi_\theta(\mathbf{a}_i|\mathbf{s}_i)/\pi_{old}(\mathbf{a}_i|\mathbf{s}_i)$. In extensive testing, Schulman et al. (2017) concluded that the clipped surrogate loss with the clipping hyperparameter $\epsilon = 0.2$ is the recommended choice. This is also the version that we use in this paper in all PPO vs. PPO-CMA comparisons.

Comparing Figure 1 and Figure 2 shows that PPO is more stable than vanilla policy gradient, but can result in somewhat similar premature shrinkage of exploration variance as in the case of only using

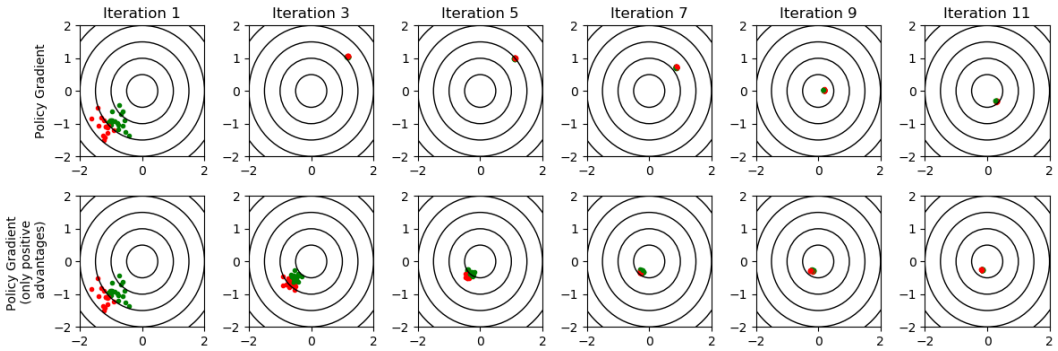

Figure 2: Comparing policy gradient variants in the didactic example of Figure 1, when doing multiple minibatch updates with the data from each iteration. Actions with positive advantage estimates are shown in green, and negative advantages in red. Top: Basic policy gradient is highly unstable. Bottom: Using only positive advantage actions, policy gradient is stable but converges prematurely.

the positive advantages for policy gradient. Although Schulman et al. (2017) demonstrated good results in MuJoCo problems with a Gaussian policy, the most impressive Roboschool results did not adapt the variance through gradient updates. Instead, the policy network only output the Gaussian mean and a linearly decaying variance with manually tuned decay rate was used.

3.3 Covariance Matrix Adaptation Evolution Strategy

The Evolution Strategies (ES) community has worked on similar variance adaptation and Gaussian exploration problems for decades, culminating in the widely used CMA-ES optimization method and its recent variants (Hansen & Ostermeier (2001); Hansen (2006); Beyer & Sendhoff (2017); Loshchilov et al. (2017)). CMA-ES is a black-box optimization method for finding a parameter vector $\mathbf{x}$ that maximizes some objective or fitness function $f(\mathbf{x})$. The CMA-ES core iteration is summarized in Algorithm 2.

Algorithm 2 High-level summary of CMA-ES

1: **for** iteration=1,2,... **do**
2: Draw samples $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{C})$.
3: Evaluate $f(\mathbf{x}_i)$
4: Sort the samples based on $f(\mathbf{x}_i)$ and compute weights $\mathbf{w}_i$ based on the ranks such that best samples have highest weights.
5: Update $\boldsymbol{\mu}$ and $\mathbf{C}$ using the samples and weights.
6: **end for**

Using the default CMA-ES parameters, the weights of the worst 50% of samples are set to 0, i.e., those samples are pruned and have no effect. The mean $\boldsymbol{\mu}$ is updated as a weighted average of the samples, but the covariance update is more involved. Although CMA-ES converges to a single optimum and the sampling distribution is unimodal, the method performs remarkably well on multimodal and/or noisy functions such as Rastrigin if using enough samples per iteration (Hansen & Kern (2004)). For full details of the update process, the reader is referred to Hansen's excellent tutorial (Hansen (2016)).

Usually, CMA-ES and other ES variants are applied to policy optimization in the form of *neuroevolution*, i.e., directly optimizing the policy network parameters, $\mathbf{x} \equiv \theta$, with $f(\mathbf{x})$ evaluated as the sum of rewards over one or more simulation episodes (Wang et al. (2010); Geijtenbeek et al. (2013); Such et al. (2017)). This is both a benefit and a drawback; neuroevolution is simple to implement and requires no critic network, but on the other hand, the sum of rewards may not be very informative in guiding the optimization. Especially in long episodes, some explored actions may be good and should be learned, but the sum may still be low. In this paper, we are interested in whether ideas from CMA-ES could improve the sampling of actions in RL, using $\mathbf{x} \equiv \mathbf{a}$.

Instead of a single action optimization task, RL is in effect solving multiple action optimization tasks in parallel, one for each possible state. The accumulation of rewards over time further complicates matters. However, we can directly apply CMA-ES to our "stateless" didactic example with $f(\mathbf{x}) \equiv r(\mathbf{a})$, illustrated in Figure 3. Comparing this to Figure 2 reveals two key insights:

- CMA-ES prunes samples based on sorted fitness values and discards the worst half. This is visually and conceptually similar to computing the policy gradient only using positive advantages, i.e., *pruning action samples based on advantage sign.* Thus, the advantage function can be thought analogous to the fitness function, although in policy optimization with a continuous state space, one can't directly enumerate, sort, and prune the actions and advantages for each state.

- Unlike PPO and policy gradient with only positive advantages, CMA-ES avoids the premature variance shrinkage problem. Instead, it increases the variance on objective function slopes to speed up progress and shrinks the variance once the optimum has been found. This can be thought as a form of momentum that acts indirectly through the exploration variance.

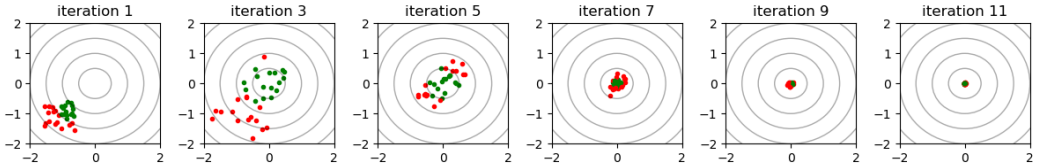

Figure 3: Our didactic example solved by CMA-ES using $\mathbf{x} \equiv \mathbf{a}$ and $f(\mathbf{x}) \equiv r(\mathbf{a})$. Red denotes pruned samples that have zero weights in updating the exploration distribution. CMA-ES expands the exploration variance while progressing on an objective function slope, and shrinks the variance when reaching the optimum.

Considering the above, a good policy optimization approach might be to only use actions with positive advantages, if one could just borrow the variance adaptation techniques of CMA-ES. In the next section, we show that this is indeed possible, resulting in our proposed PPO-CMA algorithm.

4 PPO-CMA

Our proposed PPO-CMA algorithm is summarized in Algorithm 3. Source code is available at GitHub[3]. PPO-CMA is simple to implement, only requiring minor changes to vanilla PPO, defined on Algorithm 3 lines 8-10:

- We use the standard policy gradient loss in Equation 6.

- We train only on actions with positive advantage estimates, i.e., we implement CMA-ES -style pruning of actions, but based on advantage sign instead of sorted fitness function values. As explained in Section 3.1, this also ensures that the policy stays in the proximity of the data without an explicit PPO-style KL-divergence penalty or the clipped surrogate loss function.

- We use separate neural networks for policy mean and variance, trained in separate passes. We also maintain a history of training data over H iterations, used for training the variance network. As elaborated below, these result in the CMA-ES -style variance adaptation behavior shown in Figure 1.

[3] `https://github.com/ppocma/ppocma`

Algorithm 3 PPO-CMA

for iteration=1,2,... **do**
  while iteration simulation budget N not exceeded **do**
    Reset the simulation to a (random) initial state
    Run agent on policy π_θ for T timesteps or until a terminal state
  end while
  Train critic network for K epochs using the experience from the current iteration
  Estimate advantages A^π using GAE (Schulman et al. (2015b))
  Clip negative advantages to zero, $A^\pi \leftarrow \max(A^\pi, 0)$
  Train policy variance for K epochs using experience from past H iterations and Eq. 6
  Train policy mean for K epochs using the experience from this iteration and Eq. 6
end for

4.1 Two-step updating of mean and covariance

Superficially, the core iteration loop of CMA-ES is similar to other optimization approaches with recursive sampling and distribution fitting such as the Cross-Entropy Method (De Boer et al. (2005)) and Estimation of Multivariate Normal Algorithm (EMNA) (Larrañaga & Lozano (2001)). However, there is a crucial difference: *CMA-ES first updates the covariance and only then updates the mean* (Hansen (2016)). This has the effect of elongating the exploration distribution along the best search directions instead of shrinking the variance prematurely, as shown in Figure 4. This has also been shown to correspond to a natural gradient update of the exploration distribution (Ollivier et al. (2017)).

In PPO-CMA, we implement the two-step update by using separate neural networks for the mean and variance. We first train the variance network while keeping the mean fixed, and then vice versa, using the Gaussian policy gradient loss in Equation 6.

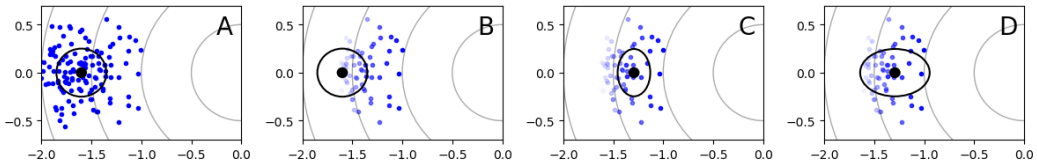

Figure 4: The difference between joint and separate updating of mean and covariance, denoted by the black dot and ellipse. A) sampling, B) pruning and weighting of samples based on fitness, C) EMNA-style update, i.e., estimating mean and covariance based on weighted samples, D) CMA-ES update, where covariance is estimated before updating the mean.

4.2 Evolution path

CMA-ES also features the so-called evolution path heuristic, where a component $\alpha \mathbf{p}^{(i)} \mathbf{p}^{(i)T}$ is added to the covariance, where α is a scalar, the (i) superscript denotes iteration index, and $\mathbf{p}$ is the evolution path (Hansen (2016)):

$$\mathbf{p}^{(i)} = \beta_0 \mathbf{p}^{(i-1)} + \beta_1 (\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}^{(i-1)}). \tag{7}$$

Although the exact computation of the default β_0 and β_1 multipliers is rather involved, Equation 7 essentially amounts to first-order low-pass filtering of the steps taken by the distribution mean between iterations. When CMA-ES progresses along a continuous slope of the fitness landscape, $||\mathbf{p}||$ is large, and the covariance is elongated and exploration is increased along the progress direction. Near convergence, when CMA-ES zigzags around the optimum in a random walk, $||\mathbf{p}|| \approx 0$ and the evolution path heuristic has no effect.

In policy optimization, policy mean and variance depend on the agent state and should be adapted differently at different parts of the state space. For some states, the policy may already be nearly

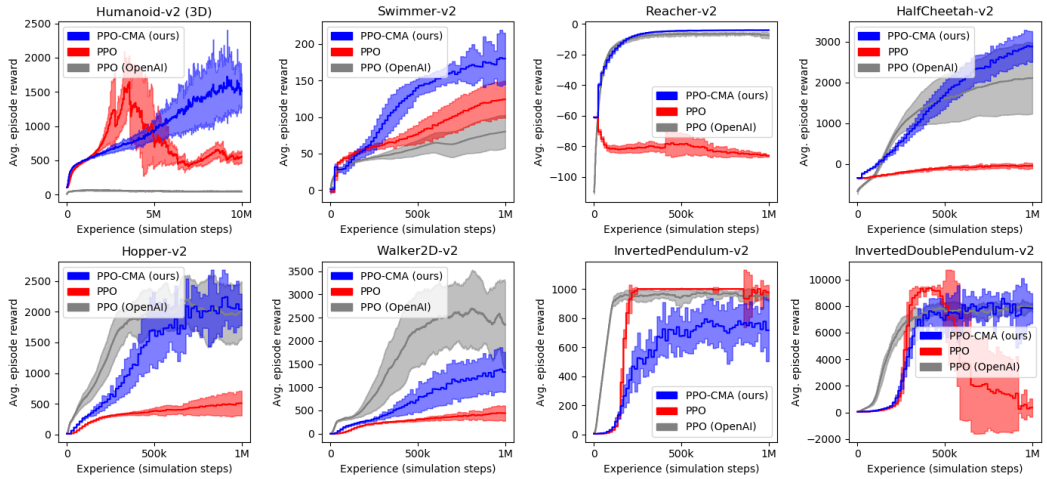

Figure 5: Means and standard deviations of training curves in OpenAI Gym MuJoCo tasks. The humanoid results are from 5 runs and others from 10 runs. Red denotes our vanilla PPO implementation with the same hyperparameters as PPO-CMA, providing a controlled comparison of the effect of algorithm changes. Gray denotes OpenAI's baseline PPO implementation using their default hyperparameters and training scripts for these MuJoCo tasks.

converged while exploration is still needed for other states. Thus, in principle, one would need yet another neural network to maintain and approximate a state-dependent $\mathbf{p}(\mathbf{s})$, which might be unstable. Fortunately, one can achieve a similar effect by simply keeping a history of H iterations of data and sampling the variance training minibatches from the history instead of only the latest data. Same as the original evolution path heuristic, this elongates the variance for a given state if the mean is moving in a consistent direction.

Appendix B provides an empirical analysis of the effect of different H values. Values larger than 1 minimize the probability of low-reward outliers, and PPO-CMA does not appear to be highly sensitive to the exact choice.

5 Evaluation

We evaluate PPO-CMA using the 7-task MuJoCo-1M benchmark used by Schulman et al. (2017) plus the more difficult 3D humanoid locomotion task. We performed the following experiments:

- The blue and red curves in Figure 5 visualize the results using PPO-CMA and PPO with exactly same implementation and hyperparameters. This ensures that the differences are due to algorithm changes instead of hyperparameters. PPO-CMA performs better in 7 out of 8 tasks. The hyperparameters are detailed in Appendix B.

- To test whether our results generalize to other hyperparameter settings, we generated 50 random hyperparameter combinations as detailed in Appendix C, and used them to train the 2D walker using both PPO and PPO-CMA. After 1M simulation steps, PPO-CMA and PPO achieved mean episode reward of 393 and 161, respectively. The standard deviations were 208 and 81. A Welch t-test indicates that the difference is statistically significant ($p < 0.001$). Although the individual training runs have high reward variance, PPO-CMA wins in 47 out of 50 runs.

- Reinforcement learning algorithms are notoriously sensitive to implementation details, and even closely related algorithms might have different optimal parameters. Thus, to compare our results against a finetuned PPO implementation, Figure 5 also shows the performance

of OpenAI's baseline PPO [4]. In this case, PPO-CMA produces better results in 4 out of 8 tasks.

Overall, PPO-CMA often progresses somewhat slower but appears less likely to diverge or stop improving.

Appendix A provides further comparisons that also visualize the variance adaptation behavior. To augment the visual comparison, Appendix C presents statistical significance testing results.

6 Related work

In addition to PPO, our work is closely related to Continuous Actor Critic Learning Automaton (CACLA) (van Hasselt & Wiering (2007)). Similar to PPO-CMA, CACLA uses the sign of the advantage estimate – in their case the TD-residual – in the updates, shifting policy mean towards actions with positive sign. The paper also observes that using actions with negative advantages can have an adverse effect. In light of our discussion of how only using positive advantage actions guarantees that the policy stays in the proximity of the collected experience, CACLA can be viewed as an early Proximal Policy Optimization approach, which we extend with CMA-ES style variance adaptation.

Although PPO is based on a traditional policy gradient formulation, there is a line of research suggesting that the so-called natural gradient can be more efficient in optimization (Amari (1998); Wierstra et al. (2008); Ollivier et al. (2017)). Through the connection between CMA-ES and natural gradient, PPO-CMA is related to various natural gradient RL methods (Kakade (2002); Peters & Schaal (2008); Wu et al. (2017)), although the evolution path heuristic is not motivated from the natural gradient perspective (Ollivier et al. (2017)).

PPO represents on-policy RL methods, i.e., experience is assumed to be collected on-policy and thus must be discarded after the policy is updated. Theoretically, off-policy RL should allow better sample efficiency through the reuse of old experience, often implemented using an experience replay buffer, introduced by Lin (1993) and recently brought back to fashion (e.g, Mnih et al. (2015); Lillicrap et al. (2015); Schaul et al. (2015); Wang et al. (2016)). PPO-CMA can be considered as a hybrid method, since the policy mean is updated using on-policy experience, but the history or replay buffer for the variance update also includes older off-policy experience.

In addition to neuroevolution (discussed in Section 3.3), CMA-ES has been applied to continuous control in the form of trajectory optimization. In this case, one searches for a sequence of optimal controls given an initial state, and CMA-ES and other sampling-based approaches (Al Borno et al. (2013); Hämäläinen et al. (2014; 2015); Liu et al. (2016)) complement variants of Differential Dynamic Programming, where the optimization utilizes gradient information (Tassa et al. (2012; 2014)). Although trajectory optimization approaches have demonstrated impressive results with complex humanoid characters, they require more computing resources in run-time.

Finally, it should be noted that PPO-CMA falls in the domain of model-free reinforcement learning approaches. In contrast, there are several model-based methods that learn approximate models of the simulation dynamics and use the models for policy optimization, potentially requiring less simulated or real experience. Both ES and RL approaches can be used for the optimization (Chatzilygeroudis et al. (2018)). Model-based algorithms are an active area of research, with recent work demonstrating excellent results in limited MuJoCo benchmarks (Chua et al. (2018)), but model-free approaches still dominate the most complex continuous problems such as humanoid movement.

For a more in-depth review of continuous control policy optimization methods the reader is referred to Sigaud & Stulp (2018) or the older but mathematically more detailed Deisenroth et al. (2013).

7 Conclusion

Proximal Policy Optimization (PPO) is a simple, powerful, and widely used model-free reinforcement learning approach. However, we have shown that in continous control with Gaussian policy,

[4] `https://github.com/openai/baselines`, we use the default parameters of run_mujoco.py, run_humanoid.py

PPO can adapt the exploration variance in an unreliable manner, which explains how PPO may converge slowly or get stuck in local optima.

As a solution to the variance adaptation problem, we have proposed the PPO-CMA algorithm that implements two-step update and evolution path heuristics inspired by the CMA-ES black-box optimization method. This results in an improved PPO version that is simple to implement but beats the original method in many MuJoCo tasks, is less prone to getting stuck in local optima, and provides a new link between RL and ES approaches to policy optimization.

Additionally, our simulations in Appendix A show that PPO's premature convergence can also be prevented with simple clipping of the policy variance or using the entropy loss term proposed by Schulman et al. (2017). However, the clipping limit or entropy loss weight needs delicate finetuning and both techniques also result in a more noisy final policy.

On a more general level, one can draw the following conclusions and algorithm design insights from our work:

- To understand the differences, similarities, and problems of policy optimization methods, it can be useful to visualize "stateless" special cases such as the one in Figure 1. PPO's problems were not at all clear to us until we created the visualizations, originally meant for teaching.
- It can be possible to adapt a black-box optimization method like CMA-ES for policy optimization by 1) treating the advantage function as the fitness function, 2) approximating sorting and pruning operations as clipping advantage values below a limit to zero (sorting is not possible because with a continuous state space, one can't enumerate all the actions sampled for a given state), and 3) considering algorithm state such as exploration mean and variance as functions of agent state instead of plain variables, and encoding them as weights of multiple neural networks to allow independent updates. Essentially, the fitness-advantage analogue allows one to think that multiple parallel CMA-ES optimizations of actions are being done for different agent states, and the neural networks store and interpolate algorithm state as a function of agent state.
- Our work highlights how gradient-based RL can have problems if the gradient affects exploration in subsequent iterations, which is the case with Gaussian policies. The fundamental problem is that a gradient that causes an increase in the expected rewards does not quarantee further increases in subsequent iterations. Instead, one should adapt exploration such that it provides good results over the whole training process. We have shown that one way to achieve this can be through an approximation of CMA-ES variance adaptation.

Thinking of the advantage function not as a way to compute accurate gradient but as a tool for pruning the learned actions also begs the question of what other pruning methods might be applicable. Presently, we are experimenting with continuous control Monte Carlo Tree Search methods (e.g., Rajamäki & Hämäläinen (2018)) for better exploration and pruning.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Mazen Al Borno, Martin De Lasa, and Aaron Hertzmann. Trajectory optimization for full-body movements with complex contacts. *IEEE transactions on visualization and computer graphics*, 19(8):1405–1414, 2013.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Hans-Georg Beyer and Bernhard Sendhoff. Simplify your covariance matrix adaptation evolution strategy. *IEEE Transactions on Evolutionary Computation*, 21(5):746–759, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, Freek Stulp, Sylvain Calinon, and Jean-Baptiste Mouret. A survey on policy search algorithms for learning robot controllers in a handful of trials. *arXiv preprint arXiv:1807.02303*, 2018.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics*, 32(6), 2013.

Danijar Hafner, James Davidson, and Vincent Vanhoucke. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878*, 2017.

Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. Online Motion Synthesis Using Sequential Monte Carlo. *ACM Transactions on Graphics*, 33(4):51, 2014.

Perttu Hämäläinen, Joose Rajamäki, and C Karen Liu. Online Control of Simulated Humanoids Using Particle Belief Propagation. *ACM Transactions on Graphics*, 34(4):81, 2015.

Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pp. 75–102. Springer, 2006.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Nikolaus Hansen and Stefan Kern. Evaluating the cma evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pp. 282–291. Springer, 2004.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

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

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pp. 5574–5584, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Pedro Larrañaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long Ji Lin. Scaling up reinforcement learning for robot control. In *Proc. 10th Int. Conf. on Machine Learning*, pp. 182–189, 1993.

Libin Liu, Michiel van de Panne, and KangKang Yin. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Transactions on Graphics*, 35(3), 2016.

Ilya Loshchilov, Tobias Glasmachers, and Hans-Georg Beyer. Limited-memory matrix adaptation for large scale black-box optimization. *arXiv preprint arXiv:1705.06693*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. Discovering and synthesizing humanoid climbing movements. *ACM Transactions on Graphics (TOG)*, 36(4):43, 2017.

Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. Information-geometric optimization algorithms: A unifying picture via invariance principles. *Journal of Machine Learning Research*, 18(18):1–65, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *arXiv preprint arXiv:1804.02717*, 2018.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

Joose Rajamäki and Perttu Hämäläinen. Augmenting sampling based controllers with machine learning. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pp. 11:1–11:9, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5091-4. doi: 10.1145/3099564.3099579. URL `http://doi.acm.org/10.1145/3099564.3099579`.

Joose Julius Rajamäki and Perttu Hämäläinen. Continuous control monte carlo tree search informed by multiple experts. *IEEE transactions on visualization and computer graphics*, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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.

Olivier Sigaud and Freek Stulp. Policy search in continuous action domains: an overview. *arXiv preprint arXiv:1803.04706*, 2018.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4906–4913. IEEE, 2012.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-Limited Differential Dynamic Programming. In *IEEE International Conference on Robotics and Automation*, pp. 1168–1175. IEEE, 2014.

Hado van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pp. 272–279. IEEE, 2007.

Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, pp. 73. ACM, 2010.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 3381–3387. IEEE, 2008.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288, 2017.

A Further analysis of PPO's variance adaptation

A.1 Variance adaptation in Hopper-V2

To further inspect the variance adaptation problem, we use the Hopper-V2 MuJoCo environment, where the monopedal agent gets rewards for traveling forward while keeping the upper part of the body upright. There is a local optimum that tends to attract policy optimization: Instead of discovering a stable gait, the agent may greedily lunge forward and fall, either right at episode start or after one or a few steps.

Figure 6 shows the training trajectories from 20 independent runs with different random seeds. The figure shows both the growth of rewards and adaptation of variance, the latter plotted as the policy standard deviation averaged over all episodes and explored actions of each iteration. The figure also includes results of the same using our proposed PPO-CMA algorithm with the same hyperparameters.

Figure 6 uses red color to denote the training runs where the policy failed to escape the local optimum. Looking at the reward and variance curves together, one sees that PPO produces worse results than PPO-CMA and PPO's average standard deviation decreases faster. Some of PPO's failure cases also exhibit erratic sharp changes in standard deviation. PPO-CMA has no clear low-reward outliers adapts variance robustly and consistently.

Many PPO implementations, including OpenAI's baseline, use some form of automatic normalization of state observations, as some MuJoCo environments have observation variables with both very large and very small scales. Figure 7 shows 20 Hopper training curves with the same normalization we use in our evaluation in Section 5. PPO-CMA is more robust to the normalization.

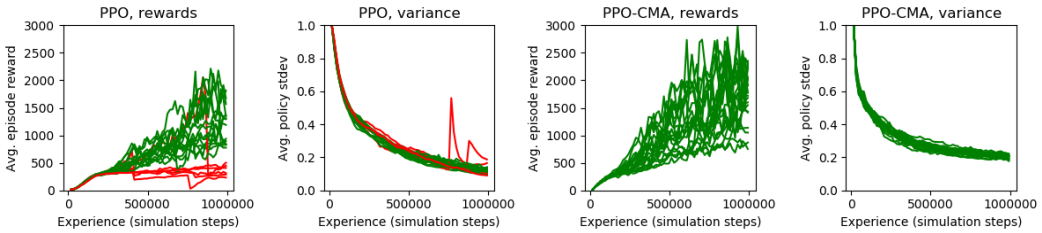

Figure 6: Comparing PPO and PPO-CMA in 20 training runs with the Hopper-V2 environment. Red trajectories denote clear failure cases where the reward plateaus and the agent only falls forward or takes one or a few steps. PPO's variance decreases faster and there is also some instability with variance peaking suddenly, associated with a decrease of rewards. PPO-CMA has no similar plateaus and adapts variance robustly and consistently.

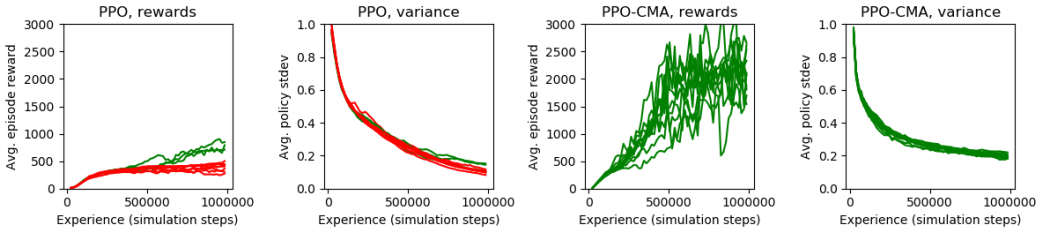

Figure 7: Same as Figure 6 but with the automatic state observation normalization described in Appendix B. PPO-CMA is more robust to the normalization. PPO failure cases are associated with lower variance.

A.2 Clipping and initializing the policy network

Similar to previous work, we use a fully connected policy network with a linear output layer and treat the variance output as log variance $\mathbf{v} = \log(\mathbf{c})$. In our initial tests with PPO, we ran into numerical precision errors which could be prevented by soft-clipping the mean as $\boldsymbol{\mu}_{clipped} = \mathbf{a}_{min} + (\mathbf{a}_{max} - \mathbf{a}_{min}) \otimes \sigma(\boldsymbol{\mu})$, where $\mathbf{a}_{max}$ and $\mathbf{a}_{min}$ are the action space limits. Similarly, we clip the log variance as $\mathbf{v}_{clipped} = \mathbf{v}_{min} + (\mathbf{v}_{max} - \mathbf{v}_{min}) \otimes \sigma(\mathbf{v})$, where $\mathbf{v}_{min}$ is a lower limit parameter, and $\mathbf{v}_{max} = 2\log(\mathbf{a}_{max} - \mathbf{a}_{min})$.

To ensure a good initialization, we pretrain the policy in supervised manner with randomly sampled observation vectors and a fixed target output $\mathbf{v}_{clipped} = 2\log(0.5(\mathbf{a}_{max} - \mathbf{a}_{min}))$ and $\boldsymbol{\mu}_{clipped} = 0.5(\mathbf{a}_{max} + \mathbf{a}_{min})$. The rationale behind this choice is that the initial exploration Gaussian should cover the whole action space but the variance should be lower than the upper clipping limit to prevent zero gradients. Without the pretraining, nothing quarantees sufficient exploration for all observed states.

One might think that premature convergence could be prevented simply by increasing the lower clipping limit. However, Figure 8 shows how a fairly high lower limit – considering the valid action range (-1,1) – is needed for the policy's standard deviation in order to ensure that all training runs escape the local optimum. This is not ideal, as it causes considerable motor noise that does not vanish as training progresses. Excessive motor noise is undesireable especially in animation and robotics applications. Finetuning the limit is also tedious, as to large values rapidly lead to worse results.

A.3 Entropy loss

The original PPO paper (Schulman et al. (2017)) discusses adding an entropy loss term to penalize low variance and prevent premature convergence, although their recommended settings for continuous control tasks do not use the entropy loss and the effect of the loss weight was not empirically investigated. Figure 8 shows how the entropy loss works similarly to the lower clipping limit, i.e., increasing the entropy loss weight helps to mimimize getting stuck in the local optimum. Too large values cause a rapid decrease in average performance and the entropy loss also results in a more noisy final policy. To mitigate this, some PPO implementations such as the one in Unity Machine Learning Agents framework anneal the entropy loss weight to zero during training; however, this adds the cost of finetuning even more hyperparameters. Ideally, one would like the variance adaptation to be both efficient and automatic.

If one wants to use PPO with variance clipping or entropy loss instead of PPO-CMA, our recommendation is to try variance clipping, as it results in better results in Figure 8. Figure 9 also shows how the entropy loss results in less stable variance curves.

B Hyperparameters and implementation details

This section describes details of our PPO-CMA and PPO implementations. OpenAI's baseline PPO was used in Section 5 without any modifications.

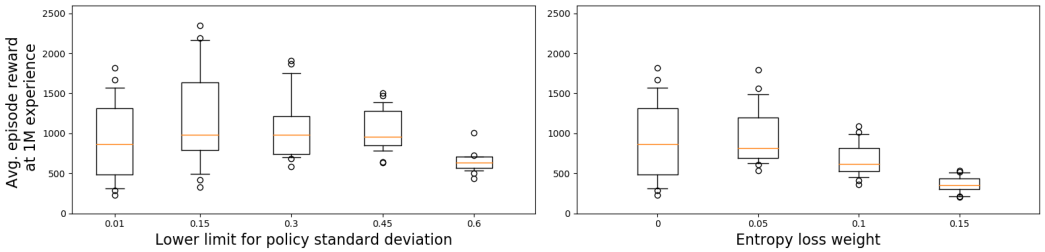

Figure 8: Boxplot of PPO results of 20 training runs of Hopper-V2 with different entropy loss weights and lower clipping limits for policy's standard deviation. The plots are from the last iteration where a limit of 1M total simulation steps was reached. The upper standard deviation limit is 1 in all plots and the entropy loss weight plots use a lower clipping limit of 0.01. Large values of either parameter can help escaping the local optimum of simply falling forward or taking only 1 or few steps (rewards below 1000), but at the same time, too large values impair average and best-case performance.

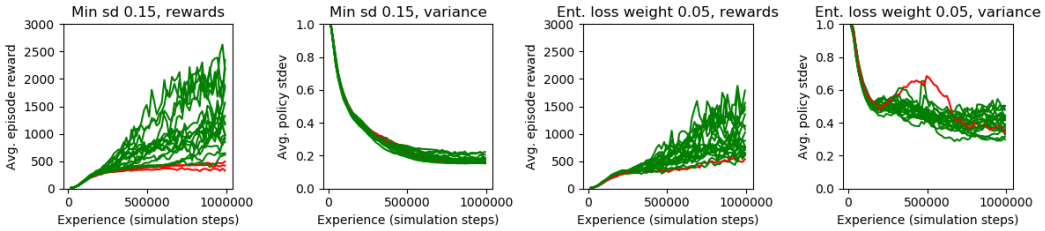

Figure 9: Comparing the effect of PPO variance clipping and entropy loss. Even with a fairly low weight, the entropy loss can lead to worse results and cause unstable increases in variance that yield low rewards (the red trajectory in the rightmost images).

We use the policy network output clipping described in Section A.2 with a lower standard deviation limit of 0.01. Thus, the clipping only ensures numerical precision but has little effect on convergence, as illustrated in Figure 8. We use the clipping because without it, we ran into numerical precision problems in PPO (but not with PPO-CMA) in environments such as the Walker2d-V2. The clipping is not necessary for PPO-CMA in our experience, but we still use with both algorithms it to ensure a controlled and fair comparison. Following Schulman's original PPO code, we also use episode time as an extra feature for the critic network to help minimize the value function prediction variance arising from episode termination at the environment time limit. Note that as the feature augmentation is not done for the policy, this has no effect on the usability of the training results.

Table 1 lists all our hyperparameters.

Hyperparameter	*Value*
Iteration simulation budget (N)	16000 (32000 for humanoid)
Training epochs per iteration (K)	20
Variance history length (H)	3 (7 for humanoid)
Minibatch size	256
Learning rate	3e-4
Network width	128 (64 for humanoid)
Num. hidden layers	2
Activation function	Leaky ReLU
Action repeat	2
Critic loss	L1

Table 1: Hyperparameters used in our PPO and PPO-CMA implementation

We use the same network architecture for all neural networks. Action repeat of 2 means that the policy network is only queried for every other simulation step and the same action is used for two steps. This speeds up training. N is specified in simulation steps. H is used only for PPO-CMA. Figure 5 is generated assuming early termination to reduce variance, i.e., for each training run, the graphs use the best scoring iteration's results so far.

The choice of $H = 3$ gives slightly better results than $H = 7$ in environments other than the 3D humanoid. The humanoid also needs a larger N to learn; the default value of 16000 results in very noisy episode rewards that quickly plateau instead of climbing steadily. This is in line with CMA-ES, which is said to be quasi-parameter-free; one mainly needs to increase the iteration sampling budget for high-dimensional and difficult optimization problems.

Figure 10 shows the effect of different H in the Hopper task. Values $H > 1$ remove low-reward outliers, but PPO-CMA does not appear to be highly sensitive to the exact choice. Comparing figures 8 and 10 suggests that the variance clipping limit, entropy loss weight and history length H all behave similarly in that one of them has to be large enough to produce good results. The crucial difference is that setting H to a value larger than the required one does not cause a rapid decrease in performance; thus, we find H easier to adjust.

We use L1 critic loss as it seems to make both PPO and PPO-CMA less sensitive to the reward scaling. For better tolerance to varying state observation scales, we use an automatic normalization scheme where observation variable j is scaled by $k_j^{(i)} = min\left(k_j^{(i-1)}, 1/\left(\rho_j + \kappa\right)\right)$, where $\kappa = 0.001$ and ρ_j is the root mean square of the variable over all iterations so far. This way, large observations are scaled down but the scaling does not constantly keep adapting as training progresses. OpenAI's baseline PPO normalizes observations based on running mean and standard deviation, but we have found our normalization slightly more stable.

B.1 On finetuning PPO-CMA and PPO

In our experience, the iteration simulation budget is the main parameter to adjust in PPO-CMA; the algorithm can make large updates and a large budget may be needed to reduce noise of the updates in high-dimensional and/or difficult problems. In addition, it may be beneficial to also experiment with different values for H. In such finetuning, it is useful to visualize full learning curve distributions like in Figure 10 instead of only plotting means and standard deviations, in order to see whether there are low-reward outliers where learning gets stuck in local optima. If there are, increasing H may help.

In light of our experiments, it is reasonable to ask why OpenAI's baseline PPO gives better results than our PPO implementation in most tasks. Our understanding is that this is due to multiple implementation details and hyperparameters that regularize and slow down the variance updates. In contrast to our settings, OpenAI's baseline PPO uses a small simulation budget of $N = 2048$ that yields many small updates, heavily regularized by the following in addition to the clipped surrogate loss:

- Training only for $K = 10$ epochs, which together with the small N amounts to significantly less minibatch updates per iteration.
- Using heavy gradient norm clipping with a clipping threshold of 0.5.
- Using smooth and saturating $\tanh$ activation functions in the neural networks, which slow down learning and result in smooth function approximations without sharp discontinuities.
- Using a smaller neural network with 64 units per layer.

In addition to the above, some PPO implementations such as the one in Unity Machine Learning Agents framework use a global diagonal covariance independent of state, which adapts slowly as a compromise over actions of all states. As a downside, the global covariance is at least theoretically less powerful because the agent may encounter both states where it should already converge and states where it should keep exploring.

C STATISTICAL SIGNIFICANCE TESTING DETAILS

To supplement the visual comparison of Figure 5, Table 2 compares PPO-CMA with OpenAI's baseline PPO using two-sided Welch t-tests.

Environment	*PPO-CMA*	*PPO (OpenAI baseline)*	*p-value*
Hopper-v2	2035.1	2023.3	0.9479
Walker2d-v2	1329.1	**2345.3**	0.0104
HalfCheetah-v2	**2874.7**	2103.8	0.0313
Reacher-v2	**-4.1**	-7.6	0.0003
Swimmer-v2	**179.9**	80.1	0.0000
InvertedPendulum-v2	715.5	**945.5**	0.0103
InvertedDoublePendulum-v2	7836.0	7838.2	0.9955
Humanoid-v2	**1491.6**	49.8	0.0238

Table 2: Final mean episode rewards in the training runs of Figure 5, together with p-values from two-sided Welch t-tests. Bolded values indicate statistically significantly better results using the 0.05 significance level.

In section 5, we compare PPO-CMA and PPO on average, over a set of randomized hyperparameter combinations. The parameters were sampled uniformly in the ranges given in Table 3.

Hyperparameter	*Randomization range*
Action repeat	$\{1, 2\}$
Learning rate	$\left[10^{-4}, 10^{-3}\right]$
Network width	$\{16, ..., 256\}$
Activation function	*tanh* or *Leaky ReLU*
N	$\{2000, ..., 64000\}$
K	$\{1, ..., 20\}$
Minibatch size	$\{32, ..., 1024\}$

Table 3: Randomization ranges of the random hyperparameter comparison of Section 5.

D LIMITATIONS

We only implement an approximation of the full CMA-ES algorithm, because in addition to the mean and covariance of the search distribution, CMA-ES maintains several other auxiliary variables. In policy optimization, all algorithm state and auxiliary variables are functions of agent state and must be encoded as neural network weights, and CMA-ES has some delicately fine-tuned mechanisms which might be unstable with such approximations. We defer further investigations of this to future work, since even our simple approximation yields benefits over vanillla PPO.

We have also only focused on Gaussian policies. Obviously, other exploration noise distributions such as Poisson might also work. However, Gaussian exploration is common in the literature, and the success of CMA-ES shows that it results in state-of-the-art performance in many optimization tasks beyond policy optimization.

Finally, treating the advantage function as fitness function assumes that the advantage landscape does not change significantly when policy is updated. This assumption does not hold in general for $\gamma > 0$. Thus, it is perhaps surprising that PPO-CMA works as well as it does. In future work, we plan to test whether PPO-CMA is particularly effective in problems that can be modeled using $\gamma = 0$, e.g., finding an optimal billiards shot that pockets as many balls as possible, or the humanoid climbing moves of Naderi et al. (2017), with actions parameterized as control splines instead of instantaneous torques.

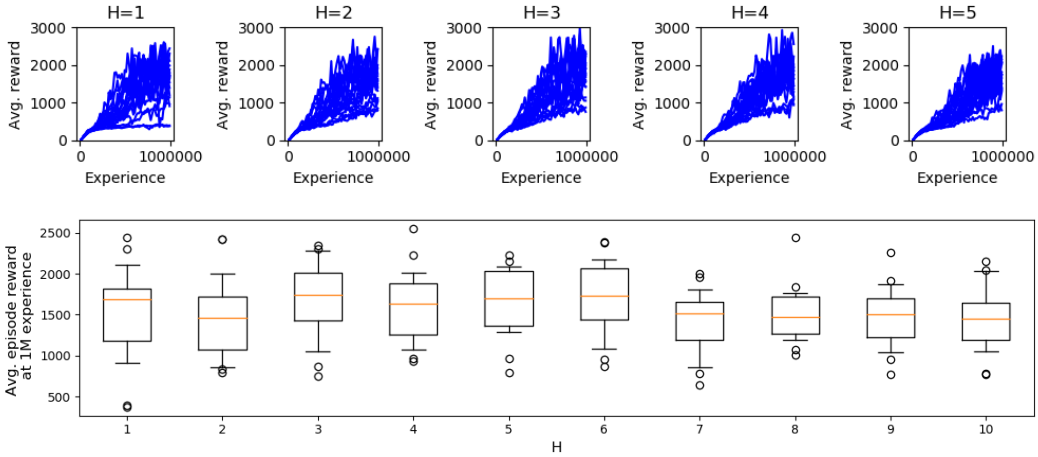

Figure 10: Training curves and final reward boxplots of PPO-CMA in 20 training runs of Hopper-V2 with different history lengths H. The orange lines show medians and the whiskers show the percentile range 10...90. With $H = 1$, there are outliers below a reward of 500, in which case the agent only lunges forward and falls without taking any steps.