PPO-CMA: Proximal Policy Optimization with Covariance Matrix Adaptation

Perttu Hämäläinen [1] **Amin Babadi** [1] **Xiaoxiao Ma** [1] **Jaakko Lehtinen** [1 2]

Abstract

Proximal Policy Optimization (PPO) is a highly popular model-free reinforcement learning (RL) approach. However, we observe that in a continuous action space, PPO can prematurely shrink the exploration variance, which leads to slow progress and may make the algorithm prone to getting stuck in local optima. Drawing inspiration from CMA-ES, a black-box evolutionary optimization method designed for robustness in similar situations, we propose PPO-CMA, a proximal policy optimization approach that adaptively expands the exploration variance to speed up progress. This can be considered as a form of action-space momentum. With only minor changes to PPO, our algorithm considerably improves performance in Roboschool continuous control benchmarks.

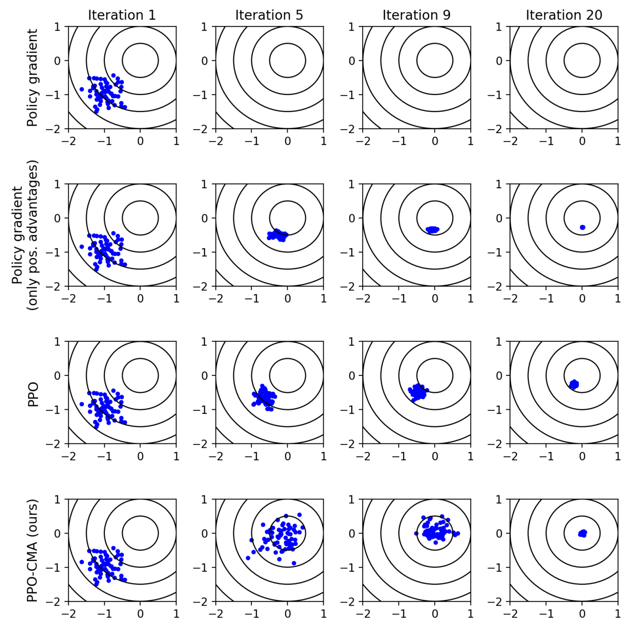

Figure 1. Comparing policy optimization methods with a simple ”stateless” quadratic objective (details in Section 2.6), when performing multiple gradient steps per iteration for improved efficiency, PPO-style. Sampled actions $\mathbf{a} \in \mathbb{R}^2$ are shown in blue. Vanilla policy gradient is unstable, which can be fixed using PPO or only positive advantages; however, the sampling/exploration variance shrinks prematurely, which leads to slow final progress. The proposed PPO-CMA method dynamically expands the variance to speed up progress, and only shrinks the variance when close to the optimum. Source code and animated visualization can be found at: `https://github.com/ppocma/ppocma`.

1. Introduction

Policy optimization with high-dimensional continuous state and action spaces is a central, long-standing problem in robotics and computer animation. In the general case, one does not have a differentiable model of the dynamics and must proceed by trial and error, i.e., try something (sample actions from an exploration distribution, e.g., a neural network policy), see what happens, and learn from the results (update the exploration distribution such that good actions become more probable). 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; 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 both during training and run-time (Hämäläinen et al., 2014; 2015; Rajamäki & Hämäläinen, 2017). PPO has later 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 (OpenAI, 2018).

[1]Department of Computer Science, Aalto University, Helsinki, Finland [2]NVIDIA. Correspondence to: Perttu Hämäläinen <perttu.hamalainen@aalto.fi>.

The key to PPO's success is mathematical and conceptual simplicity combined with excellent or at least good enough performance in a variety of problems; at the core, PPO simply takes gradient steps on the exploration distribution parameters. Stability is ensured by limiting the divergence from the old distribution. PPO has been quickly adopted as the default reinforcement learning (RL) algorithm in popular frameworks like Unity Machine Learning Agents (Juliani et al., 2018) and TensorFlow Agents (Hafner et al., 2017).

In this paper, we make the following contributions:

- We provide evidence of how PPO's exploration variance can shrink prematurely, which leads to slow progress. Figure 1 illustrates this in a simple didactic problem. For stability, PPO confines the next iteration's exploration distribution to the proximity (trust region) of the previously sampled/explored actions; if the exploration variance shrinks, the trust region and the distribution updates can grow progressively smaller in subsequent iterations. Figure 1 shows how this dynamic emerges even with a simple quadratic objective that should be easy to optimize.
- We propose PPO-CMA, a method that dynamically expands and contracts the exploration variance, inspired by the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) optimization method. Previously, CMA-ES has been applied to policy optimization in the form of *neuroevolution*, i.e., directly sampling neural network weights, which does not scale to large networks. In contrast, we show how CMA-ES principles can be used to improve the sampling of actions and implement a form of *action-space momentum* in a standard episodic RL framework. This only requires minor changes to vanilla PPO but improves performance considerably.

In the following, we first go through some preliminaries, including an overview of PPO and the didactic problem in Figure 1 that we use to visualize how the exploration variance evolves. We then proceed to analyze PPO's variance adaptation problems, how CMA-ES adapts the exploration variance, and how the core ideas of CMA-ES are implemented in PPO-CMA.

2. Preliminaries

2.1. Reinforcement Learning

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 the policy parameterized by θ, e.g., neural network weights. We focus on on-policy methods where the optimized policy also defines the exploration distribution. Executing the sampled action 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.

Both PPO and the PPO-CMA 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 the iteration simulation budget N is exhausted, θ is updated. This is summarized in Algorithm 1.

Algorithm 1 Episodic 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
```

2.2. Policy Gradient with Advantage Estimation

Policy gradient methods update policy parameters by estimating the gradient $\mathbf{g} = \nabla_\theta \mathbb{E}[\sum_t^{\infty} \gamma^t r_t]$. In practice, one often uses a compute graph framework like TensorFlow (Abadi et al., 2016) to minimize a corresponding loss function. PPO utilizes the following policy gradient loss:

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

where i denotes minibatch sample index and M is minibatch size. $A^\pi(\mathbf{s}_i, \mathbf{a}_i)$ denotes the *advantage function*, which measures the benefit of taking action $\mathbf{a}_i$ in state $\mathbf{s}_i$. Positive A^π means that the action was good and minimizing the loss function will increase the probability of sampling the same action again. Note that A^π does not directly depend on θ and thus acts as a constant when computing the gradient of Equation 1.

Same as PPO, we use Generalized Advantage Estimation (GAE) (Schulman et al., 2015b), a simple but effective way to estimate A^π using a critic network trained with the observed rewards summed over experience trajectories.

2.3. Proximal Policy Optimization

The basic idea of PPO is that one performs not just one but multiple minibatch gradient steps with the experience of each iteration. Essentially, one reuses the same data to make more progress per iteration, while stability is ensured by limiting the divergence between the old and updated policies (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 (Schulman et al., 2017) proposes two variants: 1) using an additional loss term that penalizes KL-divergence between the old and updated policies, and 2) using the so-called clipped surrogate loss function. The paper concludes that the clipped surrogate loss is the recommended choice. This is also the version that we use in this paper in all PPO vs. PPO-CMA comparisons.

2.4. PPO as Weighted Maximum Likelihood

PPO can also be interpreted as weighted maximum likelihood fitting of the policy to the sampled actions, with the advantages as the weights. This provides a crucial link to CMA-ES, which also iteratively samples from an exploration distribution and then fits the distribution to the samples weighted based on their objective function values.

Minimizing the policy gradient loss of Equation 1 corresponds to maximum likelihood fitting if one disregards the dependency of states on the preceding actions and states. This is what PPO effectively does when performing multiple gradient steps without updating the data. In this case, recalling that each action $\mathbf{a}_i$ only depends on $\mathbf{s}_i$, the joint probability density of all actions simplifies to:

$$\mathcal{P}_\theta = p_\theta(\mathbf{a}_1, ..., \mathbf{a}_N | \mathbf{s}_1, ..., \mathbf{s}_N) = \prod_{i=1}^{N} \pi_\theta(\mathbf{a}_i|\mathbf{s}_i). \tag{2}$$

If one considers the weighted case where each $\mathbf{a}_i$ is repeated in the data for w_i times, one gets:

$$\mathcal{P}_\theta = \prod_{i=1}^{N} \left[\pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\right]^{w_i}. \tag{3}$$

Weighted maximum likelihood estimation of the policy parameters θ amounts to the maximization $\theta^* = \arg\max_\theta \mathcal{P}_\theta$. Equivalently, one can minimize the loss

$$\mathcal{L}_{ML} = -\log \mathcal{P}_\theta = -\sum_{i=1}^{N} w_i \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i). \tag{4}$$

This has the same form as the policy gradient loss of Equation 1, with the weights w_i in place of the advantages.

2.5. Continuous Action Spaces

With a continuous action space, it is common to use 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 the most simple case of isotropic unit Gaussian exploration, $\mathbf{C} = \mathbf{I}$, the loss function in Equation 1 becomes:

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

Intuitively, minimizing the loss drives the policy mean towards positive-advantage actions and away from negative-advantage actions.

Following the original PPO paper, we use a diagonal covariance matrix parameterized by a vector $\mathbf{c}_\theta(\mathbf{s}) = diag(\mathbf{C}_\theta(\mathbf{s}))$. In this case, the loss becomes:

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

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

2.6. Visualizing Policy Optimization: The Didactic Problem

Throughout this paper, we make frequent use of the didactic problem in Figure 1 that allows simple visualization of how the policy evolves. In effect, we simplify policy optimization into a generic black-box optimization problem in a 2D action space, which allows plotting the distribution of actions sampled from the policy.

In the didactic problem, 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 $\mathbf{s}$ and $\mathbf{a}$ denote the first state and action of an episode. Thus, we can use $T = 1$ and focus on visualizing only the first timesteps of each episode.

Further, we use a state-agnostic $r(\mathbf{s}, \mathbf{a}) = r(\mathbf{a}) = -\mathbf{a}^T\mathbf{a}$. Thus, we have a simple quadratic optimization problem and it is enough to only visualize the action space. The optimal policy Gaussian has zero mean and variance. As everything is agnostic of agent state, the policy network can receive an arbitrary constant as its input.

3. Problem Analysis and Visualization

To motivate PPO-CMA, this section discusses and visualizes two problems inherent to advantage-based policy gradient and PPO, and how CMA-ES addresses the problems:

1. Instability caused by negative advantages when performing multiple minibatch gradient steps within each iteration to speed up convergence (Figure 2).
2. Risk of prematurely shrinking exploration variance and slow convergence when the policy updates are repeated for multiple iterations (Figure 1). It should be noted that in the RL literature, it is common to prove monotonic improvement for each iteration, but this does not quarantee efficient exploration over multiple iterations.

3.1. The Instability Caused by Negative Advantages

Considering the Gaussian policy gradient loss functions in Equations 5 and 6, it is important to note a fundamental problem: *Actions with a negative advantages may cause instability* when performing multiple minibatch gradient steps in PPO style, as each step drives the policy Gaussian further away from the negative-advantage actions.

The problem is visualized in Figure 2 (top row). The policy diverges, gravitating away from the negative-advantage actions.

3.2. Using Only Positive-Advantage Actions: Risk of Premature Convergence

From the weighted maximum likelihood perspective of Section 2.4, negative advantages (weights) are not necessarily meaningful. Indeed, simply discarding negative-advantage actions prevents the divergence and makes the policy converge to the positive-advantage actions, as illustrated on the second row of Figure 2. On the other hand, nothing prevents the exploration variance from shrinking prematurely when the procedure is repeated over multiple iterations, as illustrated in Figure 1.

3.3. PPO: Similar to Using Only Positive-Advantage Actions

As visualized in Figures 1 and 2, the clipped surrogate loss of PPO works qualitatively similar to only using the positive-advantage actions: Divergence is prevented, but the exploration variance can shrink prematurely when performing the updates over multiple iterations.

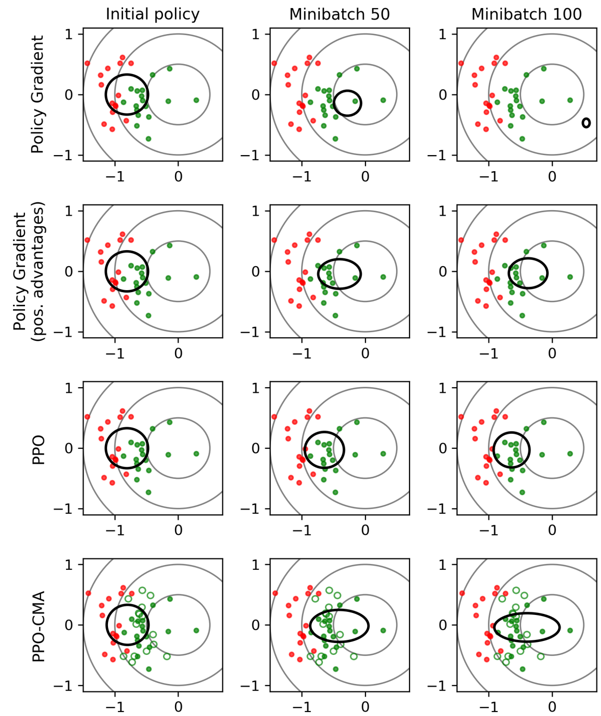

Figure 2. Policy evolution over the minibatch gradient steps of a single iteration in our didactic problem. Figure 1 shows how the same methods perform over multiple iterations. The black ellipses denote the policy mean and standard deviation according to which actions are sampled/explored. Positive-advantage actions are shown in green, negative advantages in red. The green non-filled circles show the negative-advantage actions converted to positive ones (Section 4.2). Vanilla policy gradient diverges, whereas using only the positive-advantage actions makes the policy converge to approximate the actions. PPO limits the update before convergence or divergence. PPO-CMA expands the variance in the progress direction, improving exploration in subsequent iterations.

Note that 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. Thus, our observation of variance adaptation issues complements their work instead of contradicting it.

3.4. How CMA-ES Solves the Problems

CMA-ES addresses the instability and variance adaptation problems above using a combination of three techniques:

1. When fitting the sampling distribution to weighted samples, only positive weights are used (Section 3.4.3).
2. The so-called rank-μ update elongates the exploration variance in the progress direction, making premature convergence less likely (Section 3.4.4).
3. An evolution path heuristic further expands the variance when steady progress is being made along an objective function slope (Section 3.4.5). This can be considered as a form of sampling-based momentum.

Below, we overview these key ideas, providing a foundation for Section 4, which adapts the ideas for episodic RL. We also briefly discuss why CMA-ES, as a black-box optimization method, is not applicable as such to RL.

3.4.1. CMA-ES Algorithm Summary

Using Gaussian exploration in sampling-based optimization has a long history in the Evolution Strategies (ES) literature, 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**

Although there is no convergence guarantee, CMA-ES performs remarkably well on multimodal and/or noisy functions if using enough samples per iteration (Hansen & Kern, 2004). It has also been shown that the CMA-ES mean and covariance updates can be rigorously derived from an information geometric trust region perspective (Abdolmaleki et al., 2017). For full details of the update rules, the reader is referred to Hansen's excellent tutorial (Hansen, 2016).

3.4.2. Why Not Directly Apply CMA-ES?

CMA-ES and other ES variants are usually 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, optimizing large policy networks with millions of parameters can be very computationally expensive.

In this paper, we are interested in whether ideas from CMA-ES could improve the sampling of actions in RL, using $\mathbf{x} \equiv \mathbf{a}$. This poses a much lower-dimensional problem, as even complex agents like simulated humanoids typically only have a few dozen action variables.

CMA-ES is not directly applicable to RL, as instead of a single action optimization task, RL is in effect solving multiple action optimization tasks in parallel, one for each possible state. With a continuous state space, one cannot enumerate the samples for each state, which means that the sorting operation of Algorithm 2 is not feasible. The accumulation of rewards over time further complicates matters. Fortunately, the features below can be easily implemented or approximated, as explained in Section 4.

3.4.3. Sample pruning and weights

Using the default CMA-ES parameters, the weights of the worst 50% of samples are set to 0, i.e., samples below median fitness are pruned and have no effect. The exploration mean $\boldsymbol{\mu}$ is updated in maximum likelihood manner to weighted average of the samples. Because CMA-ES uses non-negative weights, the maximum likelihood update does not diverge from the sampled actions.

Intuitively, *this is similar to policy gradient with only positive advantages (Section 3.2)*, if one considers the advantage function as analogous to the fitness $f(\mathbf{x})$, and implements the pruning based on mean instead of median fitness.

3.4.4. The rank-μ update

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: in the so-called Rank-μ update, *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 3. This has also been shown to correspond to a natural gradient update of the exploration distribution (Ollivier et al., 2017).

3.4.5. Evolution path heuristic

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,

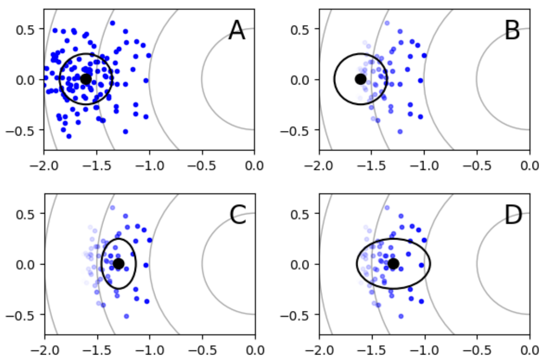

Figure 3. The difference between joint and separate updating of mean and covariance, denoted by the black dot and ellipse. A) sampling, B) pruning and weighting of samples based on fitness, C) EMNA-style update, i.e., estimating mean and covariance based on weighted samples, D) CMA-ES rank-μ update, where covariance is estimated before updating the mean. This elongates the variance in the progress direction, improving exploration in the next iteration.

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.

4. PPO-CMA

Building on the previous section, we can now describe our PPO-CMA method, summarized in Algorithm 3. Source code is available at GitHub[1]. PPO-CMA is simple to implement, only requiring the following changes to PPO:

- Instead of the clipped surrogate loss, we use the standard policy gradient loss in Equation 6 and train only on actions with positive advantage estimates to ensure stability, as motivated in Sections 3.2 and 3.4.3. However, as setting negative advantages to zero discards information, we also propose a mirroring technique for converting negative-advantage actions to positive ones (Section 4.2).

[1] https://github.com/ppocma/ppocma

- We implement the rank-μ update (Section 3.4.4) using separate neural networks for policy mean and variance, such that the variance can be updated before updating the mean.

- We maintain a history of training data over H iterations, used for training the variance network. This approximates the CMA-ES evolution path heuristic, as explained in Section 4.1.

Together, these features result in the PPO-CMA variance adaptation behavior shown in Figures 1 and 2. Despite the differences to original PPO, we still consider PPO-CMA a proximal policy optimization method, as using only positive advantages ensures that the policy does not diverge from the sampled actions and the proximity of the old policy.

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 minibatches using the experience from the current iteration
    Estimate advantages A^π using GAE (Schulman et al., 2015b)
    Clip negative advantages to zero, A^π ← max(A^π, 0) or convert them to positive ones (Section 4.2)
    Train policy variance for K minibatches using experience from past H iterations and Eq. 6
    Train policy mean for K minibatches using the experience from this iteration and Eq. 6
end for
```

4.1. Approximating the Evolution Path Heuristic

We approximate the CMA-ES evolution path heuristic (Section 3.4.5) by keeping a history of H iterations of data and sampling the variance training minibatches from the history instead of only the latest data. Similar to the original evolution path heuristic, this elongates the variance for a given state if the mean is moving in a consistent direction. We do not implement the CMA-ES evolution path heuristic directly, because this would need yet another neural network to maintain and approximate a state-dependent $\mathbf{p}(\mathbf{s})$. Similar to exploration mean and variance, $\mathbf{p}$ is a CMA-ES algorithm state variable; in policy optimization, such variables become functions of agent state and need to be encoded as neural network weights.

4.2. Mirroring Negative-Advantage Actions

Disregarding negative-advantage actions may potentially discard valuable information. We observe that assuming linearity of advantage around the current policy mean $\boldsymbol{\mu}(\mathbf{s}_i)$, it is possible to mirror negative-advantage actions about the mean to convert them to positive-advantage actions. More precisely, we set $\mathbf{a}'_i = 2\boldsymbol{\mu}(\mathbf{s}_i) - \mathbf{a}_i, A^\pi(\mathbf{a}'_i) = -A^\pi(\mathbf{a_i})\psi(\mathbf{a}_i, \mathbf{s}_i)$, where $\psi(\mathbf{a}_i, \mathbf{s}_i)$ is a Gaussian kernel (we use the same shape as the policy) that assigns less weight to actions far from the mean. This is visualized at the bottom of Figure 2. The mirroring drives the policy Gaussian away from worse than average actions, but in a way consistent with the weighted maximum likelihood estimation perspective which requires non-negative weights for stability. If the linearity assumption holds, the mirroring effectively doubles the amount of data informing the updates.

In the CMA-ES literature, a related technique is to use a negative covariance matrix update procedure (Jastrebski & Arnold, 2006; Hansen & Ros, 2010), but the technique does not improve the estimation of the mean.

5. Evaluation

A key issue in the usability of an RL method is sensitivity to hyperparameters. As learning complex tasks can take hours or days, finetuning hyperparameters is tedious. Thus, we conducted hyperparameter searches and an ablation study to investigate following questions:

- Can PPO-CMA produce better results than PPO without precise tuning of hyperparameters?
- Can hyperparameters optimized for simple tasks generalize to complex tasks?
- Is PPO-CMA less prone to getting stuck in local optima?

Our data indicates a positive answer to all the questions. Furthermore, our ablation study indicates that all the PPO-CMA algorithm features improve the results.

We used 9 OpenAI Gym Roboschool continuous control environments (OpenAI, 2017) for the hyperparemeter searches and the ablation study and tested generalization on the OpenAI Gym MuJoCo Humanoid (Brockman et al., 2016). Details of the hyperparameter search and additional results can be found in the appendix.

5.1. Sensitivity to Hyperparameters

Figure 4 visualizes sensitivity to key hyperparameters, i.e., iteration simulation budget N, PPO-CMA's history buffer size H, and PPO's clipping parameter ϵ that determines how much the updated policy can diverge from the old one. The figure reveals the following:

- PPO-CMA performs better with a wide range of hyperparameters, in particular with $H \geq 5$. Similar to CMA-ES, the main parameter to adjust is N. A large N makes progress more robust and less noisy. On the other hand, a large N means less iterations and possibly less progress within some total simulation budget, which shows as the lower scores for the largest N in Figure 4.
- There is a strong interaction of PPO's ϵ and N; if one is changed, the other must be also changed. This makes finetuning the parameters difficult, especially considering that PPO has the additional entropy loss weight w parameter to tune. Interestingly, the optimal parameter combination appears to be a very low ϵ together with a very low N. On the other hand, N should not be decreased below the episode time limit T. Most of the Roboschool environments use $T = 1000$.

5.2. Generalization and Scaling to Complex Problems

The 9 environments used for the hyperparameter search are all relatively simple 2D tasks. To test generalization and scaling, we ran both algorithms on the more challenging MuJoCo Humanoid-v2 environment. We used the best-performing hyperparameters of Figure 4 and also tested a larger simulation budget N. The agent is a 3D humanoid that gets rewards for forward locomotion. The results are shown in Figure 5. PPO-CMA yields clearly better results, especially with the increased N.

5.3. Local Optima

PPO's premature convergence observed in the didactic problem of Figure 1 may mean that it is more prone to getting stuck in local optima. The reward plateaus of Figure 6 give evidence of this. In the Hopper environment, the agent is a 2D monoped that gets rewards for staying upright and moving forward. There is a local optimum of making just one or a few big lunges forward and then falling, which results in the observed plateau level.

5.4. Ablation study

For easy replication and extending of research results, an algorithm should be as simple as possible. To check that PPO-CMA has no redundant features, we tested different ablated versions on the 9 Roboschool environments used for Figure 4, using 5 independent training runs with different random seeds for each environment. Table 1 shows the resulting normalized scores. The results indicate that all the proposed algorithm components improve performance.

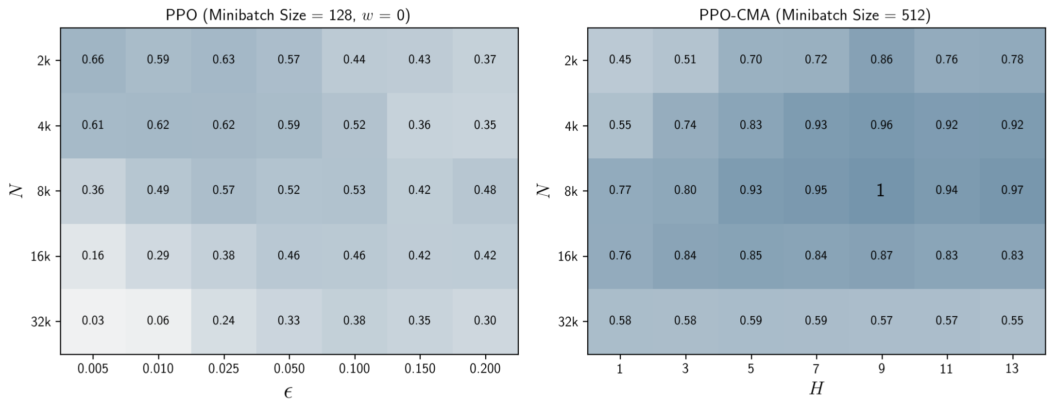

Figure 4. PPO and PPO-CMA performance as a function of key hyperparameters, using average normalized scores from 9 Roboschool environments (1 is the best observed score). The batch sizes and entropy loss weight w values are the ones that produced the best results in our hyperparameter searches. PPO-CMA performs overall better, is not as sensitive to the hyperparameter choices, and the hyperparameters can be adjusted more independently. In contrast, PPO requires careful finetuning of both the ϵ and N parameters.

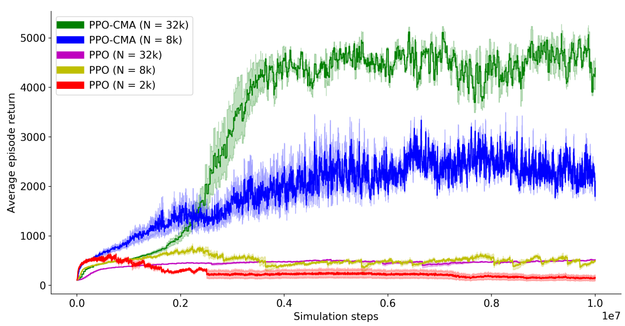

Figure 5. Comparing PPO and PPO-CMA in the MuJoCo Humanoid-v2 environment, showing means and standard deviations of training curves from 3 runs with different random seeds.

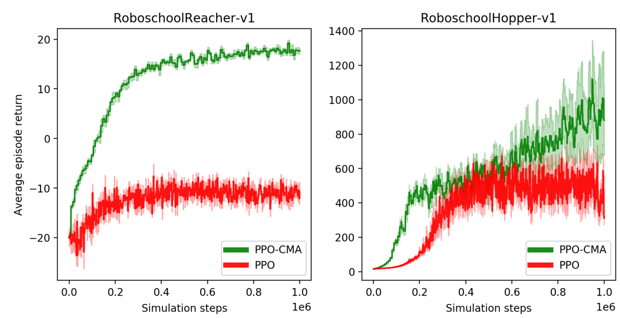

Figure 6. Some Roboschool environments clearly show how PPO's progress plateaus, indicating a tendency to get stuck in a local optimum. The training runs use the best hyperparameter combinations in Figure 4. More results are provided in the appendix.

Algorithm version	*Score*
Full PPO-CMA	1
No mirroring of negative-advantage actions	0.82
No mirroring, no evolution path heuristic	0.71
No mirroring, no evolution path, no rank-μ	0.57

Table 1. Ablation study results, showing normalized scores similar to Figure 4. Note that not using the rank-μ heuristic amounts to using a single policy network that outputs both mean and variance.

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 PPO 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 by (Mnih et al., 2015; Lillicrap et al., 2015; Schaul et al., 2015; Wang et al., 2016; Haarnoja et al., 2018). 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 3.4.2), 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; Babadi et al., 2018) complement variants of Differential Dynamic Programming (DDP), 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. Trajectory optimization has also been leveraged to inform policy search using the principle of maximum entropy control (Levine & Koltun, 2013), which leads to a Gaussian policy. Furthermore, DDP has been formulated in the terms of Gaussian distributions, which permits using CMA-ES for sampling the actions of each timestep in trajectory optimization (Rajamäki et al., 2016; Rajamäki & Hämäläinen, 2018).

PPO-CMA is perhaps most closely related to the work of (Abdolmaleki et al., 2018b;a). Maximum a posteriori Policy Optimization (MPO) (Abdolmaleki et al., 2018b) also fits the policy to the collected experience using a weighted maximum likelihood approach, but negative weights are avoided through exponentiated Q-values, based on the control-inference dualism, instead of our negative advantage mirroring. Concurrently with our work, MPO has also been extended with decoupled optimization of policy mean and variance, yielding similar variance adaptation behaviour as CMA-ES and PPO-CMA; on a quadratic objective, variance is first increased and shrinks only when close to the optimum (Abdolmaleki et al., 2018a). Together, both (Abdolmaleki et al., 2018a) and our work highlight the potential of implementing CMA-ES -style exploration in RL.

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. Limitations and Future Work

A primary limitation of our approach is that the policy mean update does not utilize off-policy data. Although PPO-CMA improves results over PPO, our MuJoCo humanoid result is not as good as what can be obtained using the state-of-the-art off-policy method Soft Actor Critic (Haarnoja et al., 2018). However, a strength of PPO-CMA is its conceptual and mathematical simplicity, and it should be possible to extend the core ideas of PPO-CMA to off-policy RL, which we are currently investigating.

Our use of off-policy data for the variance updates makes gradient estimates biased. However, we do not assume an unbiased gradient. It is well known that with any non-spherical objective function like those encountered in most practical problems, the gradient does not point towards the optimum. Thus, strictly following the gradient is not optimal. Instead, one typically uses Adam (Kingma & Ba, 2014) or some other momentum-based optimization approach. In effect, this is purposefully biased gradient descend. Our use of off-policy data approximates the CMA-ES evolution path heuristic, a form of action-space momentum, as illustrated in Figure 1. This is a novel contribution that might also be incorporated into other RL methods in future work.

Using two policy networks increases the cost of policy network training. However, this is only a minor limitation, as the total computing cost also comprises training the value function predictor network and simulating the environments to collect experience.

Finally, we did not have the resources to search over the hyperparameters of other algorithms, which is why we limit our comparison to PPO and PPO-CMA. Comparing against other algorithms with their default parameters would not be fair, as RL methods are notoriously sensitive to hyperparameter choices, including details of the neural networks (Henderson et al., 2017). Comparing against CMA-ES is also not meaningful, as the policy network has over 20k parameters; directly optimizing these with CMA-ES is prohibitively slow due to the algorithm's $\mathcal{O}(N^3)$ complexity. PPO-CMA overcomes this complexity by applying CMA-ES principles in the agent's action space, which typically has only a few dozen dimensions even for complex agents like simulated humanoids.

8. Conclusion

Proximal Policy Optimization (PPO) is a simple, powerful, and widely used model-free reinforcement learning approach. However, we have shown that PPO can prematurely shrink the exploration variance, leading to slow convergence.

As a solution to the variance adaptation problem, we have proposed the PPO-CMA algorithm that adopts the rank-μ update and evolution path heuristics of CMA-ES. Essentially, this augments PPO's Monte Carlo gradient ascend with a form of action-space momentum that dynamically expands and contracts the exploration variance, speeding up progress on objective function slopes.

PPO-CMA improves PPO results in the tested Roboschool continous control tasks while not sacrificing mathematical and conceptual simplicity. We add the separate neural networks for policy mean and variance and the H hyperparameter, but on the other hand, we do not need PPO's clipped surrogate loss function, the ϵ parameter, or the entropy loss term. Similar to CMA-ES, PPO-CMA can be said to be quasi-parameter-free; neural network architecture aside, one mainly needs to increase the iteration sampling budget N for more difficult problems.

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

- We demonstrate a new way of combining RL and ES approaches to policy optimization. Typically, ES is used for policy optimization in the form of neuroevolution, i.e., directly sampling the neural network weights. In contrast, we demonstrate how CMA-ES principles can be used to sample actions in episodic RL such that the sampling Gaussian is conditional on agent state. Conceptually, multiple parallel CMA-ES optimizations of actions are performed for different agent states, and the neural networks store and interpolate algorithm state – exploration mean and variance – as a function of agent state.
- Our work highlights the fundamental problem of policy optimization: A gradient that causes an increase in the expected rewards may not guarantee further increases in subsequent iterations due to reduced exploration. We have demonstrated that one way to solve the problem can be through an approximation of CMA-ES variance adaptation.
- 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.

Acknowledgments

This work has been supported by Academy of Finland grants 299358 and 305737.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Abdolmaleki, A., Price, B., Lau, N., Reis, L. P., and Neumann, G. Deriving and improving cma-es with information geometric trust regions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 657–664. ACM, 2017.

Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., Heess, N., and Riedmiller, M. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018a.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018b.

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

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

Babadi, A., Naderi, K., and Hämäläinen, P. Intelligent middle-level game control. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. IEEE, 2018.

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

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., and Mouret, J.-B. A survey on policy search algorithms for learning robot controllers in a handful of trials. *arXiv preprint arXiv:1807.02303*, 2018.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Deisenroth, M. P., Neumann, G., Peters, J., et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

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

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hafner, D., Davidson, J., and Vanhoucke, V. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878*, 2017.

Hämäläinen, P., Eriksson, S., Tanskanen, E., Kyrki, V., and Lehtinen, J. Online Motion Synthesis Using Sequential Monte Carlo. *ACM Transactions on Graphics*, 33(4):51, 2014.

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

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

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

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

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

Hansen, N. and Ros, R. Benchmarking a weighted negative covariance matrix update on the bbob-2010 noiseless testbed. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pp. 1673–1680. ACM, 2010.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Jastrebski, G. A. and Arnold, D. V. Improving evolution strategies through active covariance matrix adaptation. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pp. 2814–2821. IEEE, 2006.

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

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

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

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

Levine, S. and Koltun, V. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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

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

Loshchilov, I., Glasmachers, T., and Beyer, H.-G. Limited-memory matrix adaptation for large scale black-box optimization. *arXiv preprint arXiv:1705.06693*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

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

OpenAI. Roboschool. https://blog.openai.com/roboschool/, 2017.

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

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *arXiv preprint arXiv:1804.02717*, 2018.

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

Rajamäki, J. and Hämäläinen, P. 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.

Rajamäki, J. and Hämäläinen, P. Regularizing sampled differential dynamic programming. In *2018 Annual American Control Conference (ACC)*, pp. 2182–2189. IEEE, 2018.

Rajamäki, J., Naderi, K., Kyrki, V., and Hämäläinen, P. Sampled differential dynamic programming. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 1402–1409. IEEE, 2016.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015a.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Tassa, Y., Erez, T., and Todorov, E. 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.

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

van Hasselt, H. and Wiering, M. A. 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.

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

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

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

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pp. 5279–5288, 2017.

A. Implementation Details

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 $\mathbf{v}_{max} = 2\log(\mathbf{a}_{max} - \mathbf{a}_{min})$.

We use a lower standard deviation limit of 0.01. Thus, the clipping only ensures numerical precision but has little effect on convergence. 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.

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.

We train both the policy and critic networks using Adam (Kingma & Ba, 2014). Table 2 lists all our hyperparameters not included in the hyperparameter searches.

Hyperparameter	*Value*
Training minibatch steps per iteration (K)	100
Adam learning rate	0.0003
Network width	128
Number of hidden layers	2
Activation function	Leaky ReLU
Action repeat	2
Critic loss	L1

Table 2. Hyperparameters used in our PPO and PPO-CMA implementations.

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.

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.

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.

Our implementation trains the policy mean and variance networks in separate passes, keeping one network fixed while the other is trained. An alternative would be to train both networks at the same time, but cache the policy means and variances when sampling the actions, and use the cached mean for the variance network's loss function, and the cached variance for the mean network's loss.

B. Hyperparameter Search Details

The PPO vs. PPO-CMA comparison of Section 5 uses the best hyperparameter values that we found through an extensive search process. We performed the following searches:

1. A 3D search over PPO's N, ϵ, and minibatch size, using the values in Figure 4 plus minibatch sizes 128, 256, 512, 1024, 2048.

2. A 3D search over PPO's N, ϵ, and entropy loss weight $w \in \{0, 0.01, 0.05, 0.1, 0.15\}$, keeping the minibatch size at 128, which yielded the best results in the search above. Instead of a full 4D search, we chose this simplification to conserve computing resources and because the minibatch size was found to only have a minor effect on PPO's performance, as shown in Figure 7.

3. A 3D search over PPO-CMA's N, H, and minibatch size, using the values in Figure 4 plus minibatch sizes 128, 256, 512, 1024, 2048.

Each hyperparameter combination was tested using 5 independent training runs of the following 9 Roboschool environments: *Inverted pendulum, Inverted pendulum swing-up, Inverted double pendulum, Reacher, Hopper, Walker2d, HalfCheetah, Ant,* and *Pong.* In total, we performed 31500 training runs, totaling roughly two CPU years.

Each hyperparameter combination score in Figure 4 is the average of 45 normalized scores: 5 training runs with different random seeds up to 1M simulation steps, using the 9 OpenAI Gym Roboschool tasks. The scores were normalized as $R_{norm} = (R - R_{min})/(R_{max} - R_{min})$, where R is a training run's average of the non-discounted episode return $\sum_t r_t$ from the last iteration, and R_{min}, R_{max} are the minimum and maximum R of the same task over all training runs and tested hyperparameters. After the averaging, we perform a final normalization over all tested parameter combinations and algorithms such that the best combination and algorithm has score 1, i.e., PPO-CMA with minibatch size 512, $N = 8k$, and $H = 9$ in Figure 4.

We did not have the computing resources to conduct the hyperparameter searches using all the Roboschool tasks such as the Atlas humanoid robot simulation; thus, we focused on the more simple 2D tasks that could be assumed to be solved within the limit of 1M simulation steps. The results in Figure 5 indicate that the found parameters generalize beyond the simple tasks.

C. Additional Results

Figure 8 shows the training curves (mean and standard deviation of 5 runs) of all the 9 Roboschool environments used in the hyperparameter search.

Overall, PPO-CMA performs clearly better in all environments except the inverted pendulum, where initial progress is slower.

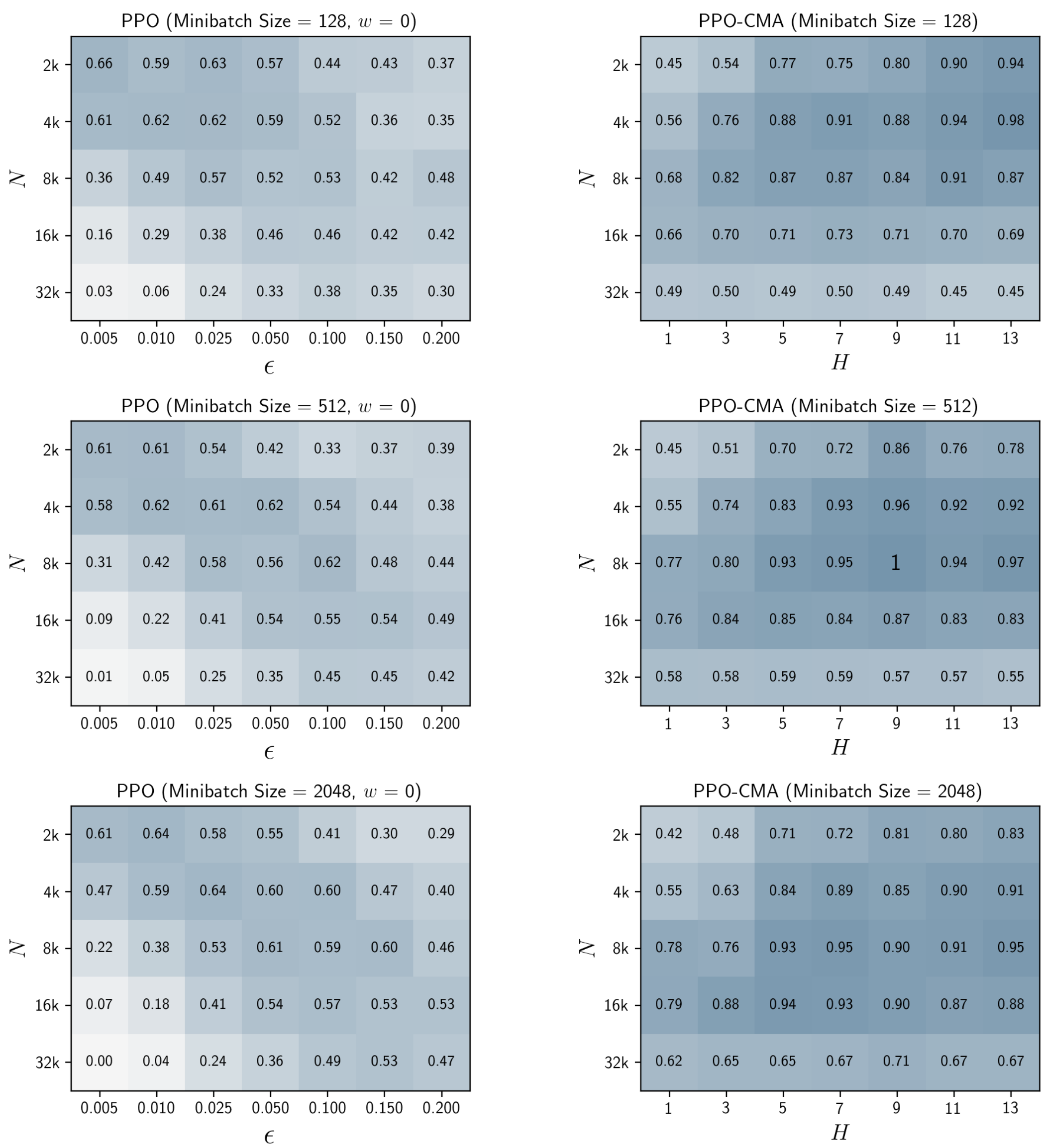

Figure 7. Slices of 3D hyperparameter search spaces, comprising minibatch size, N, and ϵ for PPO, and minibatch size, N, and H for PPO-CMA. Minibatch size has only minor effect, and the slices with different minibatch sizes look approximately similar.

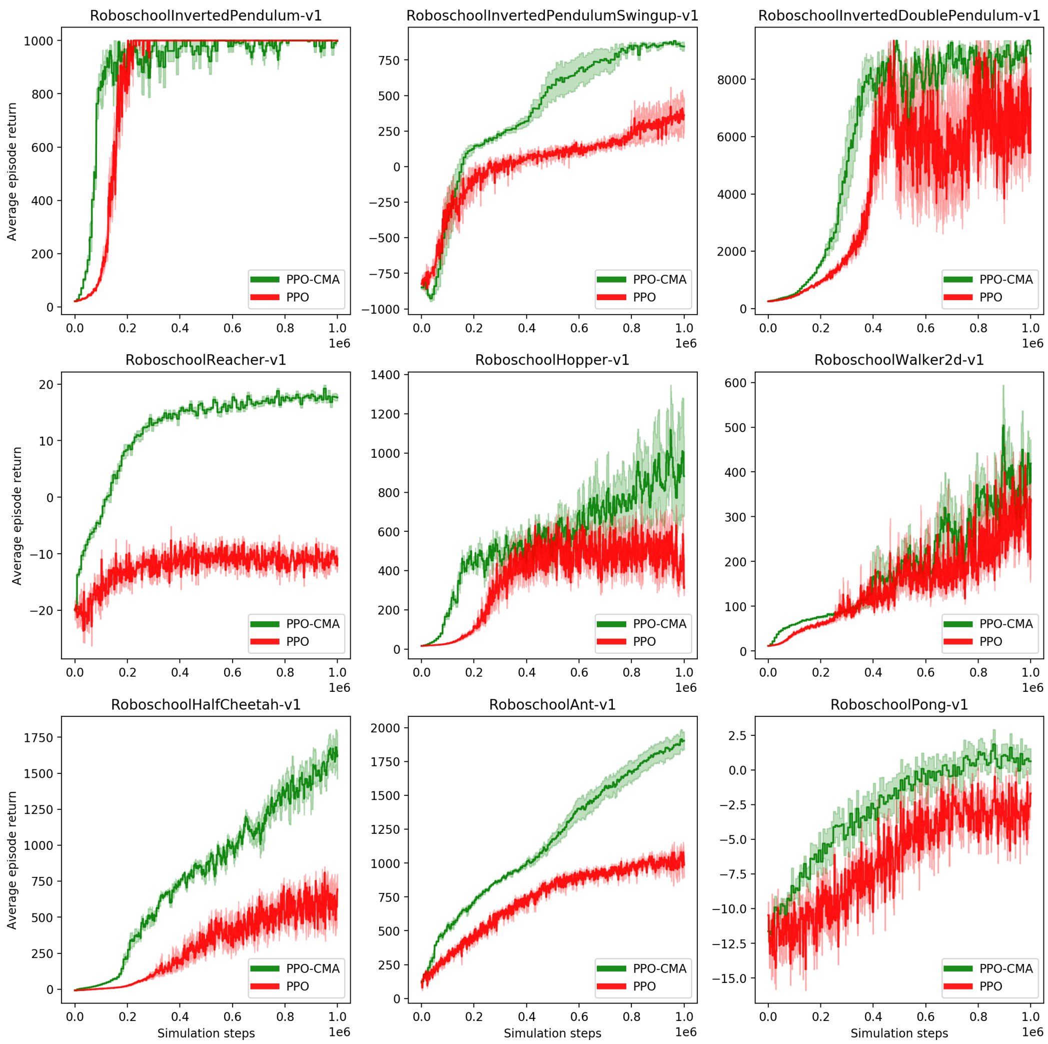

Figure 8. Training curves from the 9 Roboschool environments used in the hyperparameter search. The plots use the best hyperparameter combinations in Figure 4.