P3O: Policy-on Policy-off Policy Optimization

Rasool Fakoor*
Amazon Web Services

Pratik Chaudhari
Amazon Web Services

Alexander J. Smola
Amazon Web Services

Abstract

On-policy reinforcement learning (RL) algorithms have high sample complexity while off-policy algorithms are difficult to tune. Merging the two holds the promise to develop efficient algorithms that generalize across diverse environments. It is however challenging in practice to find suitable hyper-parameters that govern this trade off. This paper develops a simple algorithm named P3O that interleaves off-policy updates with on-policy updates. P3O uses the effective sample size between the behavior policy and the target policy to control how far they can be from each other and does not introduce any additional hyper-parameters. Extensive experiments on the Atari-2600 and MuJoCo benchmark suites show that this simple technique is highly effective in reducing the sample complexity of state-of-the-art algorithms.

1 Introduction

Reinforcement Learning (RL) refers to techniques where an agent learns a policy that optimizes a given performance metric from a sequence of interactions with an environment. There are two main types of algorithms in reinforcement learning. In the first type, called on-policy algorithms, the agent draws a batch of data using its current policy. The second type, known as off-policy algorithms, reuse data from old policies to update the current policy. Off-policy algorithms such as Deep Q-Network (Mnih et al., 2015, 2013) and Deep Deterministic Policy Gradients DDPG (Lillicrap et al., 2015) are biased (Gu et al., 2017) because behavior of past policies may be very different from that of the current policy and hence old data may not be a good candidate to inform updates of the current policy. Therefore, although off-policy algorithms are data efficient, the bias makes them unstable and difficult to tune (Fujimoto et al., 2018). On-policy algorithms do not usually incur a bias [1]; they are typically easier to tune (Schulman et al., 2017) with the caveat that since they look at each data sample only once, they have poor sample efficiency. Further, they tend to have high variance gradient estimates which necessitates a large number of online samples and highly distributed training (Ilyas et al., 2018; Mnih et al., 2016).

Efforts to combine the ease-of-use of on-policy algorithms with the sample efficiency of off-policy algorithms have been fruitful (Gu et al., 2016; O'Donoghue et al., 2016b; Wang et al., 2016; Gu et al., 2017; Nachum et al., 2017; Degris et al., 2012). These algorithms merge on-policy and off-policy updates to trade-off the variance of the former against the bias of the latter. Implementing these algorithms in practice is however challenging: RL algorithms already have a lot of hyper-parameters (Henderson et al., 2018) and such a combination further exacerbates this. This paper seeks to improve the state of affairs.

We introduce the Policy-on Policy-off Policy Optimization (P3O) algorithm in this paper. It performs gradient ascent using the gradient

$$\mathop{\mathbb{E}}_{\pi_\theta}\Big[\nabla \log \pi_\theta \, \hat{A}^{\pi_\theta}\Big] + \mathop{\mathbb{E}}_{\beta}\Big[\min(\rho, c)\, \nabla \log \pi_\theta \hat{A}^{\pi_\theta}\Big] - \lambda \nabla_\theta \mathop{\mathbb{E}}_{s\sim\beta} \mathrm{KL}\Big(\beta(\cdot|s) \,||\, \pi_\theta(\cdot|s)\Big) \tag{1}$$

where the first term is the on-policy policy gradient, the second term is the off-policy policy gradient corrected by an importance sampling (IS) ratio ρ and the third term is

*Correspondence to: Rasool Fakoor [fakoor@amazon.com] and Pratik Chaudhari [prtic@amazon.com].

[1] Note that most policy gradient implementations use the undiscounted state distribution frequencies instead of discounted distribution, these implementations are biased. However, as Thomas (2014) show being completely unbiased may even hurt performance.

a constraint that keeps the state distribution of the target policy π_θ close to that of the behavior policy β. Our key contributions are:

1. we automatically tune the IS clipping threshold c and the KL regularization coefficient λ using the normalized effective sample size (ESS), and

2. we control changes to the target policy using samples from replay buffer via an explicit Kullback-Leibler constraint.

The normalized ESS measures how efficient off-policy data is to estimate the on-policy gradient. We set

$$\lambda = 1 - \text{ESS} \quad \text{and} \quad c = \text{ESS}.$$

We show in sections 4 and A that this simple technique leads to consistently improved performance over competitive baselines on discrete action tasks from the Atari-2600 benchmark suite (Bellemare et al., 2013) and continuous action tasks from MuJoCo benchmark (Todorov et al., 2012).

2 Background

This section introduces notation and provides an overview of policy gradients, off-policy optimization techniques and covariate shift.

2.1 Notation

Let us consider an agent that interacts with the environment in discrete time steps. The policy $\pi(a|s)$ picks an action $a \in \mathcal{A}$ given the current state $s \in \mathcal{S}$. The agent receives a reward $r(s,a) \in \mathbb{R}$ after this interaction and its goal is to maximize the discounted sum of rewards $G_t = \sum_{i=t}^{\infty} \gamma^i r(s_i, a_i)$ where $\gamma \in [0,1)$ is a scalar constant that ensures that the sum is well-defined. The quantity G_t is called the return. We shorten $r(s_t, a_t)$ to r_t to simplify notation.

If the initial state s_0 is drawn from a distribution $d^0(s)$ and the agent follows the policy π thereafter, the action value function and the state only value function are given by

$$\begin{aligned} q^\pi(s_t, a_t) &= \mathop{\mathbb{E}}_{(s,a)\sim\pi}\Big[G_t | s_t, a_t\Big] \text{and} \\ v^\pi(s_t) &= \mathop{\mathbb{E}}_{a_t}\Big[q^\pi(s_t, a_t)\Big] \end{aligned} \tag{2}$$

respectively. The goal of the agent is to find a policy $\pi^* = \arg\max_\pi J(\pi)$ that maximizes the expected value of the returns where

$$J(\pi) = \mathop{\mathbb{E}}_{s\sim d^0}\Big[v^\pi(s)\Big]. \tag{3}$$

2.2 Policy gradients

We denote by π_θ, a policy that is parameterized by parameters $\theta \in \mathbb{R}^n$; these may for instance be the parameters of a deep neural network. This induces a parameterization of the state-action and state-only value functions which we denote by q^{π_θ} and v^{π_θ} respectively. Monte-Carlo policy gradient methods such as REINFORCE (Williams, 1992) solve the above optimization problem by taking gradient steps on the objective and uses the likelihood-ratio trick to estimate the policy gradient. The policy gradient of (3) is

$$\mathop{\mathbb{E}}_{s_t\sim d^{\pi_\theta}, a_t\sim\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\; q^{\pi_\theta}(s_t, a_t)\Big] \tag{4}$$

where d^{π_θ} is the unnormalized discounted state visitation frequency $d^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s)$. We will shorten the notation for the expectation $\mathbb{E}_{s_t\sim d^{\pi_\theta}, a_t\sim\pi_\theta}[\cdot]$ to $\mathbb{E}_{\pi_\theta}[\cdot]$.

Remark 1 (Baseline for variance reduction). We can estimate the policy gradient as an average of the above integrand over sample trajectories drawn using the current policy π_θ. The action value function q^{π_θ} is typically estimated using Monte-Carlo as $\hat{q}^{\pi_\theta}(s_t, a_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. Both of these entail a large variance for policy gradients (Kakade and Langford, 2002; Baxter and Bartlett, 2001) and a number of techniques exist to mitigate the variance. The most common one is to subtract a state dependent control variate (baseline) $\hat{v}^{\pi_\theta}(s)$ from $\hat{q}^{\pi_\theta}(s)$. This leads to the Monte-Carlo estimate of the advantage function (Konda and Tsitsiklis, 2000)

$$\hat{A}^{\pi_\theta}(s,a) = \hat{q}^{\pi_\theta}(s,a) - \hat{v}^{\pi_\theta}(s)$$

which is used in place of q^{π_θ} in (4). Let us note that more general state-action dependent baselines can also be used (Liu et al., 2017). We denote the baselined policy gradient integrand in short by $g(\pi_\theta) := \hat{A}^{\pi_\theta}(s,a)\, \nabla_\theta \log \pi_\theta(a|s)$ to rewrite (4) as

$$\nabla_\theta^{\text{on}} J(\pi_\theta) = \mathop{\mathbb{E}}_{\pi_\theta}\Big[g(\pi_\theta)\Big]. \tag{5}$$

2.3 Off-policy policy gradient

The policy gradient in (5) involves an expectation over data collected from the current policy π_θ. Vanilla policy gradient methods use each datum only once to update the policy; this makes then sample inefficient. A solution to this problem is to use an experience replay buffer (Lin, 1992) to store previous data and reuse these experiences to update the current policy using importance sampling. For a mini-batch of size T consisting of $\{(s_k, a_k, s'_k)\}$ for $k \le T$ drawn from the buffer, the integrand in (5)

becomes

$$\Big(\prod_{t=0}^{T} \rho(s_t, a_t)\Big) \sum_{t=0}^{T} \left(\sum_{i=0}^{T-t} \gamma^i r_{t+i}\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

where the importance sampling (IS) ratio

$$\rho(s, a) = \frac{\pi_\theta(a|s)}{\beta(a|s)} > 0 \tag{6}$$

governs the relative probability of the candidate policy π_θ with respect to the one that collected the data β.

Degris et al. (2012) employed marginal value functions to approximate the above gradient and they obtained the expression

$$\mathop{\mathbb{E}}_{s\sim d^\beta, a\sim\beta} \Big[\rho(s, a)\ \nabla_\theta \log \pi_\theta(a|s) \hat{A}^{\pi_\theta}(s, a)\Big] \tag{7}$$

for the off-policy policy gradient. Note that states are sampled from d^β which is the state distribution of β. Further, the expectation occurs using the policy β while the action value function q^{π_θ} is that of the target policy π_θ. This is important because in order to use the off-policy policy gradient above, one needs to estimate q^{π_θ}. The authors in Wang et al. (2016) estimate q^{π_θ} using the Retrace(λ) estimator (Munos et al., 2016). If π_θ and β are very different from each other, the importance ratio $\rho(s, a)$ may vary across a large magnitude. This leads to high variance if (8) is estimated using Monte-Carlo techniques. An effective way to mitigate this is to clip $\rho(s, a)$ at some threshold c. We will use this clipped importance ratio often and denote it as $\overline{\rho}_c = \min(\rho, c)$. This helps us shorten the notation for the off-policy policy gradient to

$$\nabla^{\text{off}}_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\beta} \Big[\overline{\rho}_c\ g(\pi_\theta)\Big]. \tag{8}$$

2.4 Covariate shift

In the supervised learning setting if we observe iid data from a distribution $q(x)$, minimizing the risk with respect to another distribution $p(x)$ amounts to minimizing the following weighted loss function

$$\begin{aligned} &\mathop{\mathbb{E}}_{x\sim p(x)} \mathop{\mathbb{E}}_{y|x} \left[\ell(y, \varphi(x))\right] \\ &\qquad = \mathop{\mathbb{E}}_{x\sim q(x)} \mathop{\mathbb{E}}_{y|x} \left[w(x)\ \ell(y, \varphi(x))\right]. \end{aligned} \tag{9}$$

Here y are the labels associated to draws $x \sim q(x)$ and $\ell(y, \varphi(x))$ is the loss of the predictor $\varphi(x)$. Such an importance sampling estimator is consistent but leads to increased variance because it discounts samples that have low likelihood under the new distribution $p(x)$ (Robert and Casella, 2013; Elvira et al., 2018).

Definition 2 (Effective sample size). Given a dataset $X = \{x_1, x_2, \ldots, x_N\}$ and two densities $p(x)$ and $q(x)$ with $p(x)$ being absolutely continuous with respect to $q(x)$, the effective sample size is defined as the number of samples from $p(x)$ that would provide an estimator with a performance equal to that of the importance sampling (IS) estimator in (9) with N samples (Kong, 1992). For our purposes, we will use the normalized effective sample size

$$\text{ESS} = \frac{1}{N}\ \|w(X)\|_1^2 / \|w(X)\|_2^2 \tag{10}$$

where $w(X) := [\mathrm{d}p(x_1)/\mathrm{d}q(x_1), \ldots, \mathrm{d}p(x_N)/\mathrm{d}q(x_N)]$ is a vector that consists of the Radon-Nikodym derivative of the two densities (Resnick, 2013) evaluated at the samples. This expression is a good rule of thumb and occurs, for instance, for a weighted average of Gaussian random variables (Quionero-Candela et al., 2009) or in particle filtering (Smith, 2013). We have normalized the ESS by the size of the dataset which makes $\text{ESS} \in [0, 1]$.

We can use the ESS as an indicator of the efficacy of updates to π_θ with samples drawn from the behavior policy β. If the ESS is large, the two policies predict similar actions given the state and we can confidently use data from β to update π_θ.

3 Approach

This section discusses the P3O algorithm. We first identify key characteristics of merging off-policy and on-policy updates and then discuss the details of the P3O algorithm and provide insight into its behavior using ablation experiments.

3.1 Combining on-policy and off-policy policy gradients

We can combine the on-policy update in (5) with the off-policy update in (8) after bias-correction on the former to get

$$\mathop{\mathbb{E}}_{\pi_\theta} \left[\left(1 - \frac{c}{\rho}\right)_+ g(\pi_\theta)\right] + \mathop{\mathbb{E}}_{\beta} \Big[\overline{\rho}_c\ g(\pi_\theta)\Big]; \tag{11}$$

here $(\cdot)_+ := \max(\cdot, 0)$ denotes rectification. This is similar to the off-policy actor-critic (Degris et al., 2012) and ACER gradient (Wang et al., 2016) except that the authors in Wang et al. (2016) use the Retrace(λ) estimator to estimate q^{π_θ} in (8). Note that the expectation in the second term is computed over actions that were sampled by β whereas the expectation of the first term is computed over all actions $a \in \mathcal{A}$ weighted by the probability of taking them $\pi_\theta(a|s)$. The clipping constant c in (11) controls the off-policy updates versus on-policy updates. As $c \to \infty$, ACER does a completely off-policy update

while we have a completely on-policy update as $c \to 0$. In practice, it is difficult to pick a value for c that works well for different environments as we elaborate upon in the following remark. This difficulty in choosing c is a major motivation for the present paper.

Remark 3 (How much on-policy updates does ACER do?). We would like to study the fraction of weight updates coming from on-policy data as compared to those coming from off-policy data in (11). We took a standard implementation of ACER[2] with $c = 10$ and track the on-policy part of the loss (first term in (11)) as training progresses in Fig. 1. Note that the on-policy loss is zero throughout training. This suggests that the performance of ACER (Wang et al., 2016) should be attributed pre-dominantly to off-policy updates and the Retrace(λ) estimator rather than the combination of off-policy and on-policy updates. This experiment demonstrates the importance of hyper-parameters when combining off-policy and on-policy updates.

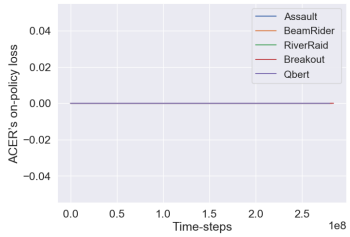


Figure 1: On-policy loss for ACER (first term in (11)) is zero all through training due to aggressive IS ratio thresholding. ACER had the highest reward from among A2C, PPO and P3O in 3 out of these 5 games (Assault, RiverRaid and BreakOut; see the Supplementary Material for more details). In spite of the on-policy loss being zero for all Atari games, ACER receives good rewards across the benchmark.

3.2 Combining on-policy and off-policy data with control variates

Another way to leverage off-policy data is to use it to learn a control variate, typically the action value function q_ω. This has been the subject of a number papers; recent ones include Q-Prop (Gu et al., 2016) which combines Bellman updates with policy gradients and Interpolated Policy Gradients (IPG) (Gu et al., 2017) which directly interpolates between on-policy and off-policy deterministic gradient, DPG and DDPG algorithms, (Silver et al., 2014; Lillicrap et al., 2016)) using a hyper-parameter. To contrast with the ACER gradient in (11), the IPG is

$$(1-\nu) \mathop{\mathbb{E}}_{\pi_\theta} \Big[g(\pi_\theta) \Big] + \nu \mathop{\mathbb{E}}_{s\sim\beta} \Big[\nabla_\theta \mathop{\mathbb{E}}_{a\sim\pi_\theta(a|s)} \{ q_\omega(s,a) \} \Big] \tag{12}$$

[2]OpenAI baselines: https://github.com/openai/baselines

where q_w is an off-policy fitted critic. Notice that since the policy π_θ is stochastic the above expression uses $\nabla_\theta \mathbb{E}_{a\sim\pi_\theta} \{q_\omega\}$ for the off-policy part instead of the DPG $\nabla_\theta q_\omega(s, \mu_\theta(s))$ for a deterministic policy $\mu_\theta(s)$. This avoids training a separate deterministic policy (unlike Q-Prop) for the off-policy part and encourages on-policy exploration and an implicit trust region update. The parameter ν explicitly controls the trade-off between the bias and the variance of off-policy and on-policy gradients respectively. However, we have found that it is difficult to pick this parameter in practice; this is also seen in the results of (Gu et al., 2017) which show sub-par performance on MuJoCo (Todorov et al., 2012) benchmarks; for instance compare these results to similar experiments in Fujimoto et al. (2018) for the Twin Delayed DDPG (TD3) algorithm.

3.3 P3O: Policy-on Policy-off Policy Optimization

Our proposed approach, named Policy-on Policy-off Policy Optimization (P3O) explicitly controls the deviation of the target policy with the behavior policy. It is given as follows

$$\begin{aligned} &\mathop{\mathbb{E}}_{\pi_\theta} \Big[g(\pi_\theta) \Big] + \mathop{\mathbb{E}}_{\beta} \Big[\overline{\rho}_c \, g(\pi_\theta) \Big] \\ &\qquad - \lambda \nabla_\theta \mathop{\mathbb{E}}_{s\sim\beta} \mathrm{KL}\Big(\beta(\cdot|s) \;||\; \pi_\theta(\cdot|s) \Big). \end{aligned} \tag{13}$$

The first term is the standard on-policy gradient, the second term in the above expression is the off-policy policy gradient with truncation of the IS ratio using a constant c while the third term allows explicit control of the deviation of the target policy π_θ from β. Note that we do not perform bias correction in the first term so it is missing the factor $\left(1 - \frac{c}{\rho}\right)_+$ from the ACER gradient (11). As we noted in Remark 3, it may be difficult to pick a value of c which keeps this factor non-zero. Note that even if the KL-term is zero, the above gradient is a biased estimate of the on-policy policy gradient. Further, note that the KL-divergence term can be rewritten as $\mathbb{E}_\beta [\log \rho]$ and therefore minimizes the importance ratio ρ over the entire replay buffer β. There are two hyper-parameters in the P3O gradient: the IS truncation threshold c and the KL regularization co-efficient λ. We use the following rationale to pick values of c and λ.

If the behavior and target policies are far from each other, we would like a large λ to push them closer. If they are too close to each other, we could perform more exploration and therefore want a smaller regularization co-efficient λ. We set

$$\lambda = 1 - \mathrm{ESS} \tag{14}$$

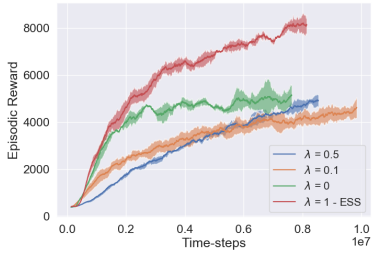
where the ESS in (10) is computed using the current mini-

batch sampled from the replay buffer β.

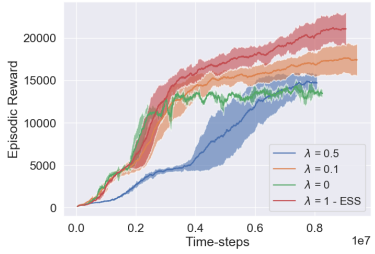
The truncation threshold c is chosen to keep the variance of the second term bounded. Smaller the c less efficient the off-policy update and larger the c higher the variance of this update. We set

$$c = \text{ESS}. \tag{15}$$

This is a very natural way to threshold the IS factor $\rho(s, a)$ because $\text{ESS} \in [0, 1]$. This adaptively chooses a trade-off between the reduced variance and the inefficiency of a small IS ratio ρ. Note that the ESS is computed on a mini-batch of transitions and their respective IS factors and hence clipping an individual $\rho(s, a)$ using the ESS tunes c automatically to the mini-batch.



(a) BeamRider



(b) Qbert

Figure 2: **Effect of λ on performance.** First, a non-zero value of λ trains much faster than without the KL regularization term because the target policy is constrained to be close to an entropic β. Second, for hard exploration games like Qbert, a smaller value $\lambda = 0.1$ works much better than $\lambda = 0.5$ while the trend is somewhat reversed for easy exploration games such as BeamRider. The ideal value of λ thus depends on the environment and is difficult to pick before-hand. Setting $\lambda = 1 - \text{ESS}$ tunes the regularization adaptively depending upon the particular mini-batch and works significantly better for easy exploration, it also leads to gains in hard exploration tasks.

The gradient of P3O in (13) is motivated from the following observation. Explicitly controlling the KL-divergence between the target and the behavior policy encourages them to have the same visitation frequencies. This is elaborated upon by Lemma 4 which follows from the time-dependent state distribution bound proved in (Schulman et al., 2015a; Kahn et al., 2017).

Lemma 4 (Gap in discounted state distributions). *The gap between the discounted state distributions d^{π_θ} and d^β is bounded as*

$$\|d^{\pi_\theta} - d^\beta\|_1 \leq \frac{2\gamma}{(1-\gamma)^2} \sqrt{\max_{s \in \mathcal{S}} \text{KL}(\beta \,||\, \pi_\theta)} \tag{16}$$

The KL-divergence penalty in (13) is directly motivated from the above lemma; we however use $\mathbb{E}_{s\sim\beta}\left[\text{KL}(\pi_\theta \,||\, \beta)\right]$ which is easier to estimate.

Remark 5 (Effect of λ). Fig. 2 shows the effect of picking a good value for λ on the training performance. We picked two games in Atari for this experiment: BeamRider which is an easy exploration task and Qbert which is a hard exploration task (Bellemare et al., 2016). As the figure and the adjoining caption shows, picking the correct value of λ is critical to achieving good sample complexity. The ideal λ also changes as the training progress because policies are highly entropic at initialization which makes exploration easier. It is difficult to tune λ using annealing schedules, this has also been mentioned by the authors in Schulman et al. (2017) in a similar context. Our choice of $\lambda = 1 - \text{ESS}$ adapts the level of regularization automatically.

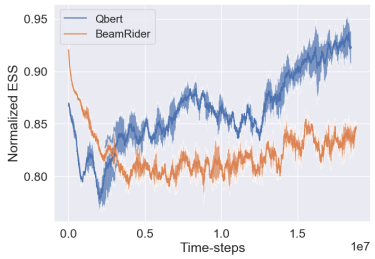
Remark 6 (P3O adapts the bias in policy gradients). There are two sources of bias in the P3O gradient. First, we do not perform correction of the on-policy term in (11). Second, the KL term further modifies the descent direction by averaging the target policy's entropy over the replay buffer. If $\rho(s, a) > c$ for all transitions in the replay buffer, the bias in the P3O update is

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\pi_\theta}\left[-\frac{c}{\rho}\nabla \log \pi_\theta \hat{A}^{\pi_\theta}\right] + \mathop{\mathbb{E}}_{s\sim\beta, a\in\mathcal{A}}\left[\lambda \nabla \log \pi_\theta(a|s)\right] \\
&= \mathop{\mathbb{E}}_{\beta}\left[-\text{ESS}\,\nabla \log \pi_\theta \hat{A}^{\pi_\theta}\right] \\
&\qquad + \mathop{\mathbb{E}}_{s\sim\beta, a\in\mathcal{A}}\left[(1-\text{ESS})\,\nabla \log \pi_\theta(a|s)\right]
\end{aligned} \tag{17}$$

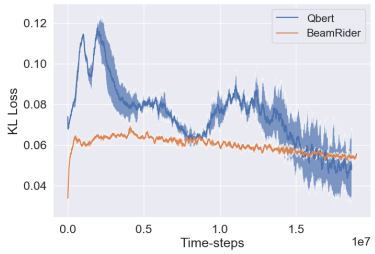
The above expression suggests a very useful feature. If the ESS is close to 1, i.e., if the target policy is close to the behavior policy, P3O is a heavily biased gradient with no entropic regularization. On the other hand, if the ESS is zero, the entire expression above evaluates to zero. The choice $c = \text{ESS}$ therefore tunes the bias in the P3O updates adaptively. Roughly speaking, if the target policy is close to the behavior policy, the algorithm is confident and moves on even with a large bias. It is difficult to control the bias coming from the behavior policy, the ESS allows us to do so naturally.

A number of implementations of RL algorithms such as Q-Prop and IPG often have subtle, unintentional biases (Tucker et al., 2018). However, the improved performance of these algorithms, as also that of P3O, suggests

that biased policy gradients might be a fruitful direction for further investigation.

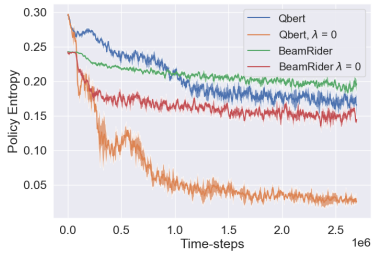


(a) ESS



(b) KL term

$$\mathbb{E}_{s\sim\beta}\left[\mathrm{KL}(\beta(\cdot|s)\,||\,\pi_\theta(\cdot|s))\right]$$



(c) Entropy of π_θ

Figure 3: **Evolution of ESS, KL penalty and the entropy of π_θ as training progresses.** Fig. 3a shows the evolution of normalized ESS. A large value of ESS indicates that the target policy π_θ is close to β in its state distribution. The ESS is about 0.85 for a large fraction of the training which suggests a good trade-off between exploration and exploitation. The KL term in Fig. 3b is relatively constant during the course of training because its coefficient λ is adapted by ESS. This enables the target policy to be exploratory while still being able to leverage off-policy data from the behavior policy. Fig. 3c shows the evolution of the entropy of π_θ normalized by the number of actions $|\mathcal{A}|$. Note that using $\lambda = 0$ results in the target policy having a smaller entropy than standard P3O. This reduces its exploratory behavior and the latter indeed achieves a higher reward as seen in Fig. 2.

3.4 Discussion of the KL penalty

The KL-divergence penalty in P3O is reminiscent of trust-region methods. These are a popular way of making monotonic improvements to the policy and avoiding premature moves, e.g., see the TRPO algorithm by Schulman et al. (2015a). The theory in TRPO suggests optimizing a surrogate objective where the hard KL divergence constraint is replaced by a penalty in the objective. In our setting, this amounts to the penalty $\lambda\ \mathbb{E}_{s\sim\beta}\left[\mathrm{KL}(\beta\,||\pi_\theta)\right]$. Note that the behavior policy β is a mixture of previous policies and this therefore amounts to a penalty that keeps π_θ close to *all* policies in the replay buffer β. This is also done by the authors in Wang et al. (2016) to stabilize the high variance of actor-critic methods.

A penalty with respect to *all* past policies slows down optimization. This can be seen abstractly as follows. For an optimization problem $x^* = \arg\min_x f(x)$, the gradient update $x^{k+1} = x^k - \alpha^k \nabla f(x^k)$ can be written as

$$x^{k+1} = \arg\min_y \left\{ \langle \nabla f(x), y\rangle + \frac{1}{2\alpha^k}\|y - x^k\|^2 \right\}$$

if the $\arg\min$ is unique; here x^k is the iterate and α^k is the step-size at the k^{th} iteration. A penalty with respect to all previous iterates $\{x^1, x^2, \ldots, x^k\}$ can be modeled as

$$x^{k+1} = \arg\min_y \left\{ \langle \nabla f(x), y\rangle + \frac{1}{2\alpha^k}\sum_{i=1}^{k}\|y - x^i\|^2 \right\} \tag{18}$$

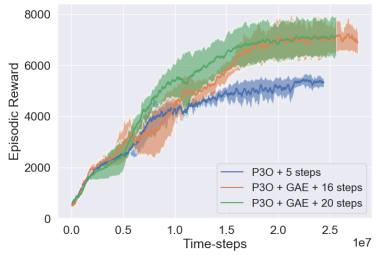
which leads to the update equation $x^{k+1} = \frac{1}{k}\sum_{i=1}^k x^i - \frac{\alpha^k}{k}\nabla f(x^k)$ which has a vanishing step-size as $k \to \infty$ if the schedule α^k is left unchanged. We would expect such a vanishing step-size of the policy updates to hurt performance.

The above observation is at odds with the performance of both ACER and P3O; see Section 4 which shows that both algorithms perform strongly on the Atari benchmark suite. However Fig. 3 helps reconcile this issue. As the target policy π_θ is trained, the entropy of the policy decreases, while older policies in the replay buffer are highly entropic and have more exploratory power. A penalty that keeps π_θ close to β encourages π_θ to explore. This exploration compensates for the decreased magnitude of the on-policy policy gradient seen in (18).

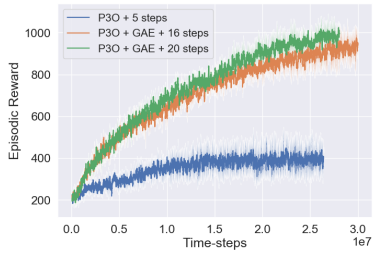
3.5 Algorithmic details

The pseudo-code for P3O is given in Algorithm 1. At each iteration, it rolls out $K = 16$ trajectories of $T = 16$ time-steps each using the current policy and appends them to the replay buffer $\mathcal{D}$. In order to be able to compute the KL-divergence term, we store the policy $\pi_\theta(\cdot|s)$ in addition to the action for all states.

P3O performs sequential updates on the on-policy data and the off-policy data. In particular, Line 5 in Algo-



(a) Ms. Pac-Man



(b) Gravitar

Figure 4: **Effect of roll-out length and GAE.** Figs. 4a and 4b show the progress of P3O with and without generalized advantage estimation. GAE leads to significant improvements in performance. The above figures also show the effect of changing the number of time-steps from the environment used in on-policy updates: longer time-horizons help in games with sparse rewards although the benefit diminishes across the suite after 20 steps.

Algorithm 1: Policy-on Policy-off Policy Optimization

Input: Policy π_θ, baseline v_ϕ, replay buffer $\mathcal{D}$

Roll out trajectories $\mathcal{b} = \{\tau_1, \tau_2, \ldots, \tau_K\}$ for T time-steps each

Compute the returns $G(\tau_k)$ and policy $\pi_\theta(\cdot|s_t;\ \tau_k)$ $\forall\, t \leq T, k \leq K$

$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{b}$

On-policy update of π_θ using $\mathcal{b}$; see (13)

$\xi \leftarrow$ Poisson(m)

for $i \leq \xi$ **do**

  $\mathcal{b}_i \leftarrow$ sample mini-batch from $\mathcal{D}$

  Estimate ESS and KL-divergence term using π_θ and stored policies $\log \mu(\cdot|s_t;\ \tau_k)$ $\forall\, t \leq T, \tau_k \in \mathcal{b}_i$

  Off-policy and KL regularizer update of π_θ using $\mathcal{b}_i$; see (13)

rithm 1 samples a Poisson random variable that governs the number of off-policy updates for each on-policy update in P3O. This is also commonly done in the literature (Wang et al., 2016). We use Generalized Advantage Estimation (GAE) (Schulman et al., 2015b) to estimate the advantage function in P3O. We have noticed significantly improved results with GAE as compared to without it, as Fig. 4 shows.

4 Experimental validation

This section presents the experimental validation of our approach. We evaluate the P3O algorithm against competitive baselines on the Atari-2600 benchmarks from OpenAI Gym (Brockman et al., 2016) and MuJoCo benchmark (Todorov et al., 2012). Our objective in this section is to demonstrate that P3O achieves performance comparable to state-of-the-art reinforcement learning algorithms with the ESS-based hyper-parameter choices motivated in Section 3. Results for MuJoCo are presented in Appendix A.

4.1 Setup

We used the same 3-layer convolutional neural network as the one used by Mnih et al. (2015) with last 4 frames returned by the environment as input to the network. We compare the P3O algorithm developed in this paper against three competitive baselines: the synchronous actor-critic architecture (A2C) of Mnih et al. (2016), proximal policy optimization (PPO) from Schulman et al. (2017) and actor-critic with experience replay (ACER) from Wang et al. (2016). The first, A2C, is a standard baseline while PPO is a completely on-policy algorithm that is robust and has demonstrated good performance on a variety of tasks including Atari and MuJoCo benchmarks. ACER combines on-policy updates with off-policy updates and is closest to P3O, as we have discussed in Section 3.

We use the same hyper-parameters as the original authors of these algorithms and implementations from OpenAI Baselines[3] in order to be consistent and comparable to existing literature. Details are provided in the Supplementary Material.

4.2 Summary of results

Table 10 shows a comparison of P3O against the three baselines averaged over all the games in the Atari-2600 benchmark. We measure progress in two ways: (i) in terms of the final reward for each algorithm averaged over the last 100 episodes after 28M time-steps (80M

[3] https://github.com/openai/baselines

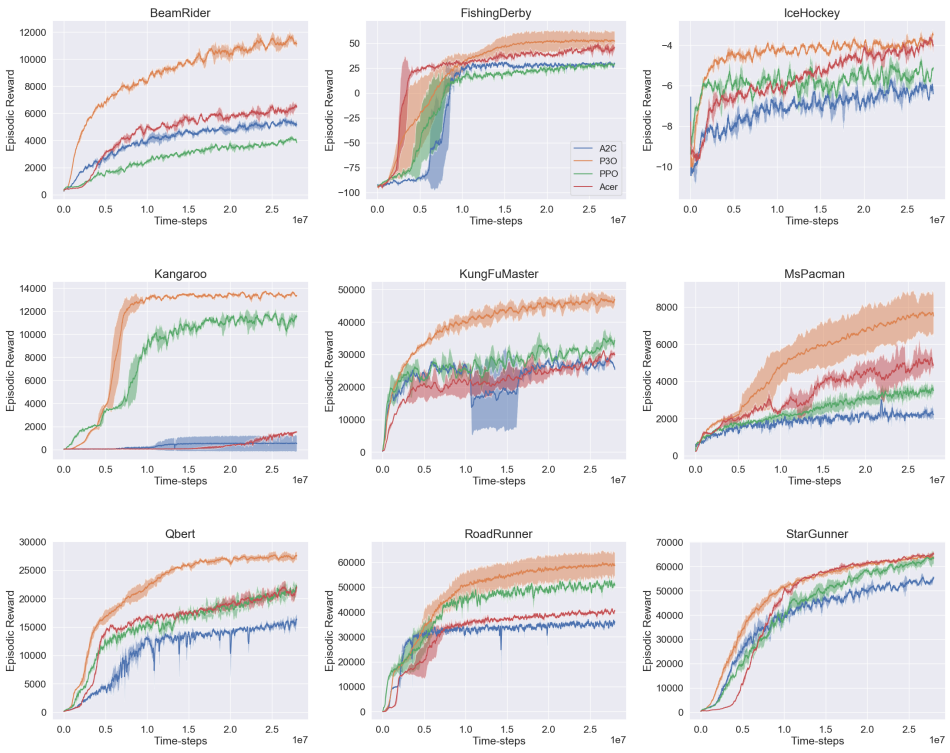


Figure 5: **Comparison of A2C (blue), ACER (red), PPO (green) and P3O (orange)** on some Atari games. Comparisons for all the 49 games are provided in the Supplementary Material.

frames of the game) and (ii) in terms of the reward at 40% training time and 80% training time averaged over 100 episodes. We average the reward over three random seeds to compute the numbers in Table 10 and we follow the evaluation protocol proposed by Machado et al. (2017) to report the results. The latter compares different algorithms in terms of their sample efficiency. These results suggest that P3O is an efficient algorithm that improves upon these competitive baselines both in terms of the final reward at the end of training and the reward obtained after a fixed number of samples. A table with the rewards for all the 49 games and algorithms is provided in the Supplementary Material. Fig. 7 shows the reward curves for some of the Atari games for all these algorithms; similar plots for all the 49 Atari games are provided in the Supplementary Material.

Table 1: Number of Atari games "won" by each algorithm measured by the average reward over 100 episodes across three random seeds.

Algorithm	Won	Won @ 40% training time	Won @ 80% training time
A2C	0	0	0
ACER	13	9	11
PPO	9	8	10
P3O	**27**	**32**	**28**

5 Related work

Our work builds upon recent techniques that combine off-policy and on-policy updates in reinforcement learning. The closest one to our approach in this paper is the ACER algorithm (Wang et al., 2016). It builds upon off-policy actor-critic method of (Degris et al., 2012); it uses the Retrace operator (Munos et al., 2016) to estimate an off-policy action value function and constrains the candidate policy to be close to the running average of past policies using a linearized KL-divergence constraint. The P3O algorithm uses a biased variant of the ACER gradient and incorporates an explicit KL penalty in the objective. We discuss this connection in detail in Section 3.

The PGQL algorithm (O'Donoghue et al., 2016a) uses an estimate of the action value function of the target policy to combine on-policy updates with those obtained from the Bellman error objective. QProp (Gu et al., 2016) learns the action value function using off-policy data which is used as a control variate for on-policy updates. The au-

thors in Gu et al. (2017) propose the interpolated policy gradient (IPG) which takes a unified view of these algorithms. It directly combines on-policy and off-policy updates using a hyper-parameter and shows that, although such updates may be biased, the bias is bounded.

A key characteristic of all the above algorithms is that they have numerous hyper-parameters which are critical to achieving good performance. For instance, the authors in Oh et al. (2018) report poor results with ACER and prioritized replay as compared to vanilla actor-critic methods. Our use of the effective sample size eliminates the clipping threshold for the importance ratio (IS) which is a key hyper-parameter.

Policy gradient algorithms with off-policy data are not new. The importance sampling ratio has been commonly used by a number of authors such Cao (2005); Levine and Koltun (2013). Effective sample size is popularly used to measure the quality of importance sampling and to restrict the search space for parameter updates (Jie and Abbeel, 2010; Peshkin and Shelton, 2002). We exploit ESS to a similar end, it is an effective way to both control the contribution of the off-policy data and the deviation of the target policy from the behavior policy. Let us note there are a number of works that learn action value functions using off-policy data, e.g.,Wang et al. (2013); Hausknecht and Stone (2016); Lehnert and Precup (2015) that achieve varying degrees of success on reinforcement learning benchmarks.

Covariate shift and effective sample size have been studied extensively in the machine learning literature; see Robert and Casella (2013); Quionero-Candela et al. (2009) for an elaborate treatment. These ideas have also been employed in reinforcement learning (Kang et al., 2007; Bang and Robins, 2005; Dudík et al., 2011). To the best of our knowledge, we are the first to use ESS for combining on-policy updates with off-policy updates.

6 Conclusion

This paper proposed Policy-on Policy-off Policy Optimization (P3O), an algorithm to combine the sample efficiency of off-policy RL algorithms with the ease of use of on-policy algorithms. We introduced a KL divergence-based constraint between the behavior policy and the target policy. We showed that the effective sample size (ESS) of the behavior policy with respect to the target policy is a remarkably effective to automatically tune both the regularization of the constraint and the importance sampling ratio threshold of the off-policy update. This combination is typically performed using hyper-parameters and is difficult to tune in practice; we showed that we can eliminate these hyper-parameters. We perform extensive ablation and large-scale experiments to show that P3O, in spite of its simplicity, consistently achieves good performance.

We have focused on discrete action spaces in this paper. The P3O algorithm can also be used for continuous action spaces without any modifications if we cache the behavior policy's logits in the replay buffer to compute the KL regularization term. Further, the P3O algorithm leverages time-varying quantities like ESS to robustly combine off-policy and on-policy updates. This suggests that understanding the dynamics of reinforcement learning algorithms might be the key to making them sample efficient and generalize to new environments.

References

Bang, H. and Robins, J. M. (2005). Doubly robust estimation in missing data and causal inference models. *Biometrics*, 61(4):962–973.

Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *NIPS*, pages 1471–1479.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *arXiv:1606.01540*.

Cao, X.-R. (2005). A basic formula for online policy gradient algorithms. *IEEE Transactions on Automatic Control*, 50(5):696–699.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv:1205.4839*.

Dudík, M., Langford, J., and Li, L. (2011). Doubly robust policy evaluation and learning. *arXiv:1103.4601*.

Elvira, V., Martino, L., and Robert, C. P. (2018). Rethinking the effective sample size. *arXiv:1809.04129*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv:1802.09477*.

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2016). Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv:1611.02247*.

Gu, S. S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., and Levine, S. (2017). Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *NIPS*, pages 3846–3855.

Hausknecht, M. and Stone, P. (2016). On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2018). Are deep policy gradient algorithms truly policy gradient algorithms? *arXiv:1811.02553*.

Jie, T. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008.

Kahn, G., Zhang, T., Levine, S., and Abbeel, P. (2017). Plato: Policy learning using adaptive trajectory optimization. In *ICRA*, pages 3342–3349. IEEE.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274.

Kang, J. D., Schafer, J. L., et al. (2007). Demystifying double robustness: A comparison of alternative strategies for estimating a population mean from incomplete data. *Statistical science*, 22(4):523–539.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In Solla, S. A., Leen, T. K., and Müller, K., editors, *NIPS*, pages 1008–1014. MIT Press.

Kong, A. (1992). A note on importance sampling using standardized weights. *Technical Report 348*.

Lehnert, L. and Precup, D. (2015). Policy gradient methods for off-policy control. *arXiv:1512.04105*.

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

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

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Liu, H., Feng, Y., Mao, Y., Zhou, D., Peng, J., and Liu, Q. (2017). Action-depedent control variates for policy optimization via stein's identity. *arXiv:1710.11198*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. (2017). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *CoRR*, abs/1709.06009.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv:1312.5602*.

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. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *NIPS*, pages 1054–1062.

Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning. In *NIPS*.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2016a). Combining policy gradient and q-learning. *arXiv:1611.01626*.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2016b). PGQ: Combining policy gradient and Q-learning. *arXiv:1611.01626*.

Oh, J., Guo, Y., Singh, S., and Lee, H. (2018). Self-imitation learning. *arXiv:1806.05635*.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience. *cs/0204043*.

Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2009). *Dataset shift in machine learning*. The MIT Press.

Resnick, S. I. (2013). *A probability path*. Springer Science & Business Media.

Robert, C. and Casella, G. (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, volume 37, pages 1889–1897.

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

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*.

Smith, A. (2013). *Sequential Monte Carlo methods in practice*. Springer Science & Business Media.

Thomas, P. (2014). Bias in natural actor-critic algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 441–448. PMLR.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., and Levine, S. (2018). The mirage of action-dependent baselines in reinforcement learning. *arXiv:1802.10031*.

Wang, Y.-H., Li, T.-H. S., and Lin, C.-J. (2013). Backward q-learning: The combination of sarsa algorithm and q-learning. *Engineering Applications of Artificial Intelligence*, 26(9):2184–2193.

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

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

A MuJoCo benchmarks

Table 2: P3O hyperparameters for MuJoCo

Hyperparameters	Value
Architecture	FC(100) - FC(100)
Learning rate	3×10^{-4}
Replay Buffer size	5×10^{3}
Number of environments	2
Number of steps per iteration	64
Entropy regularization (α)	0.0
Off policy updates per iteration (ξ)	Poisson(3)
Burn-in period	2500
Number of samples from replay buffer	15
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Advantage estimation discounting factor (τ)	0.95
Random Seeds	$\{0 \dots 9\}$

Table 3: A2C (and A2CG) hyperparameters on MuJoCo

Hyperparameters	Value
Architecture	FC(64) - FC(64)
Learning rate	13×10^{-3}
Number of environments	8
Number of steps per iteration	32
Entropy regularization (α)	0.0
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Random Seeds	$\{0 \dots 9\}$

Table 4: PPO hyperparameters on MuJoCo

Hyperparameters	Value
Architecture	FC(64) - FC(64)
Learning rate	3×10^{-4}
Number of environments	1
Number of steps per iteration	2048
Entropy regularization (α)	0.0
Number of training epochs per update	10
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Advantage estimation discounting factor (τ)	0.95
Random Seeds	$\{0 \dots 9\}$

Table 5: Performance of agents on MuJoCo continuous-control tasks after 3M time-steps of training.

Games	A2CG	A2C	PPO	P3O
HalfCheetah-v2	181.46	1907.42	2022.14	**5051.58**
Walker2d-v2	855.62	2015.15	2727.93	**3770.86**
Hopper-v2	1377.07	1708.22	2245.03	**2334.32**
Swimmer-v2	33.33	45.27	101.71	**116.87**
InvertedDoublePendulum-v2	90.09	5510.71	4750.69	**8114.05**
InvertedPendulum-v2	733.34	889.61	414.49	**985.14**
Ant-v2	−253.54	1811.29	1615.55	**4727.34**
Humanoid-v2	530.12	720.38	530.13	**2057.17**

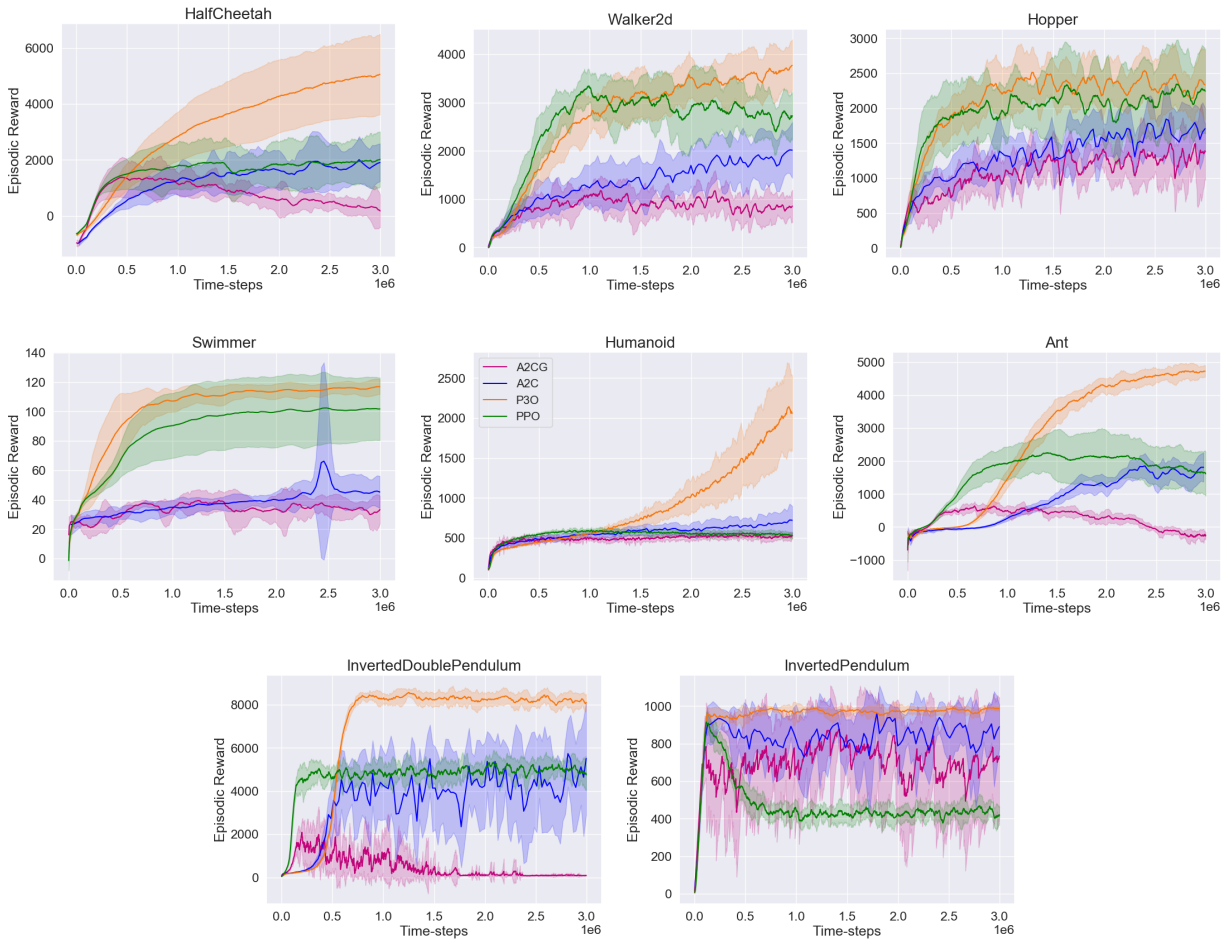


Figure 6: Comparison of A2C (blue), A2CG [A2C with GAE] (magenta), PPO (green) and P3O (orange) on 8 MuJoCo environments.

B Atari benchmarks

Tables 6 to 9 show hyper-parameters for each of the methods used in our experiments.

Table 6: A2C hyperparameters on Atari games

Hyperparameters	Value
Architecture	Conv (32-8 $\times$ 8-4) Conv (64-4 $\times$ 4-2) Conv (64-3 $\times$ 1-1) FC (512)
Learning rate	7×10^{-4}
Number of environments	16
Number of steps per iteration	5
Entropy regularization (α)	0.01
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Random Seeds	$\{0 \dots 2\}$

Table 7: ACER hyperparameters on Atari games

Hyperparameters	Value
Architecture	Same as A2C
Replay Buffer size	5×10^4
Learning rate	7×10^{-4}
Number of environments	16
Number of steps per iteration	20
Entropy regularization (α)	0.01
Number of training epochs per update	4
Discount factor (γ)	0.99
Value loss Coefficient	0.5
importance weight clipping factor	10
Gradient norm clipping coefficient	0.5
Momentum factor in the Polyak	0.99
Max KL between old & updated policy	1
Use Trust region	True
Random Seeds	$\{0 \dots 2\}$

Table 8: PPO hyperparameters on Atari games

Hyperparameters	Value
Architecture	Same as A2C
Learning rate	7×10^{-4}
Number of environments	8
Number of steps per iteration	128
Entropy regularization (α)	0.01
Number of training epochs per update	4
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Advantage estimation discounting factor (τ)	0.95
Random Seeds	$\{0 \dots 2\}$

Table 9: P3O hyperparameters on Atari games

Hyperparameters	Value
Architecture	Same as A2C
Learning rate	7×10^{-4}
Replay Buffer size	5×10^4
Number of environments	16
Number of steps per iteration	16
Entropy regularization (α)	0.01
Off policy updates per iteration (ξ)	Poisson(2)
Burn-in period	15×10^3
Number of samples from replay buffer	6
Discount factor (γ)	0.99
Value loss Coefficient	0.5
Gradient norm clipping coefficient	0.5
Advantage estimation discounting factor (τ)	0.95
Random Seeds	$\{0 \dots 2\}$

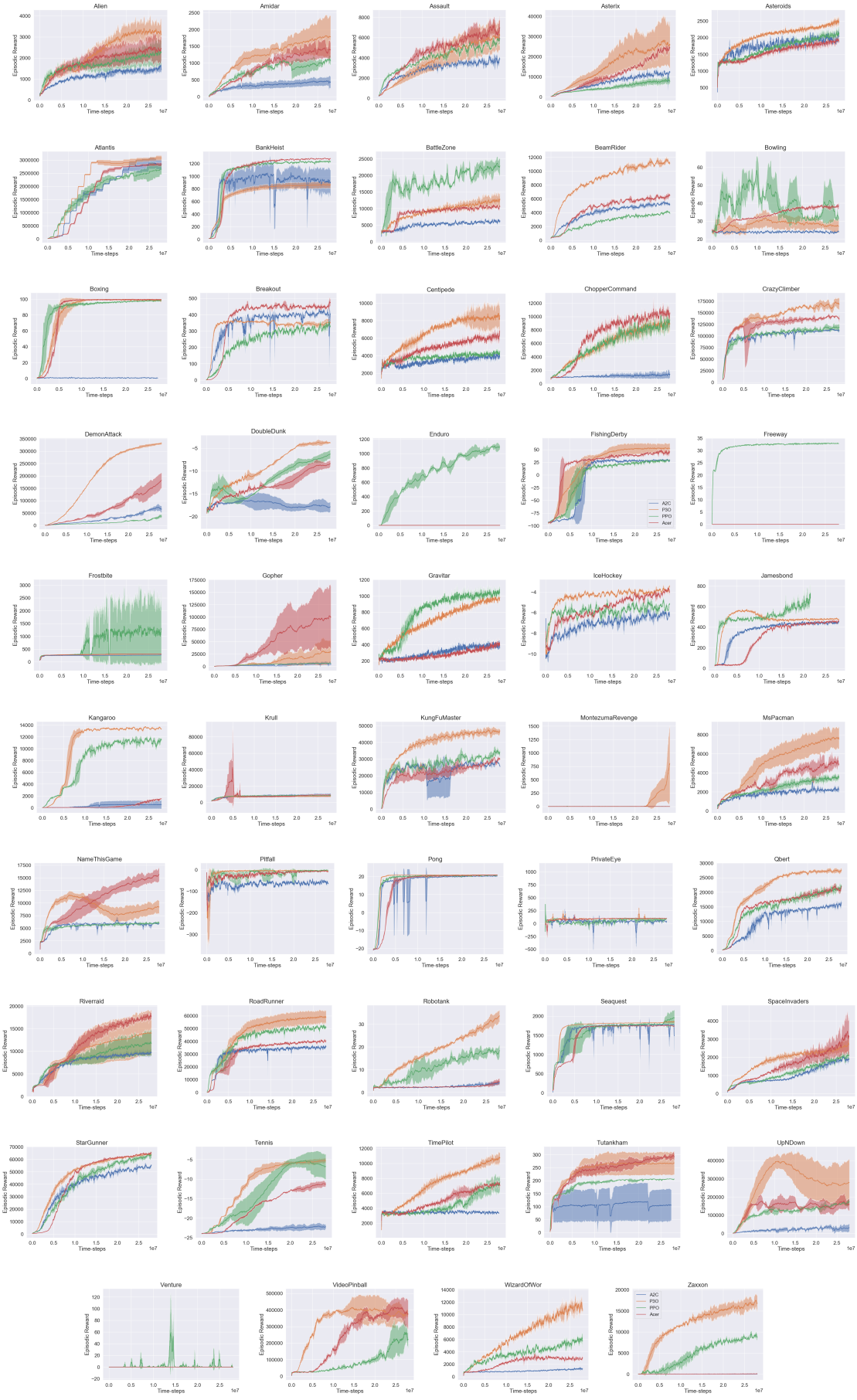


Figure 7: Comparison of A2C (blue), ACER (red), PPO (green) and P3O (orange) on all 49 Atari games.

Table 10: Performance of agents on 49 Atari-2600 games after 28M timesteps (112M frames) of training.

Games	A2C	ACER	PPO	P3O
Alien	1425.00	2436.20	2260.43	**3124.80**
Amidar	439.43	1393.24	1062.73	**1787.40**
Assault	3897.73	**6996.46**	5941.23	6222.27
Asterix	12272.50	24414.00	7574.33	**25997.00**
Asteroids	2052.27	1874.83	2147.33	**2483.30**
Atlantis	2847251.67	2832752.33	2647593.67	**3077883.00**
BankHeist	910.43	**1281.60**	1236.90	864.03
BattleZone	6250.00	10726.67	**22856.67**	12793.33
BeamRider	5149.29	6486.07	3834.01	**11163.49**
Bowling	24.19	**38.61**	31.75	27.04
Boxing	0.21	99.33	98.06	**99.44**
Breakout	403.25	**474.81**	328.80	351.81
Centipede	3722.24	6755.41	4530.21	**8615.36**
ChopperCommand	1389.67	**10376.00**	9504.33	8878.33
CrazyClimber	111418.67	136527.67	118501.00	**168115.00**
DemonAttack	65766.90	181679.27	37026.17	**331454.95**
DoubleDunk	−17.86	−8.37	−6.29	**−3.83**
Enduro	0.00	0.00	**1092.52**	0.00
FishingDerby	29.54	45.74	29.34	**52.07**
Freeway	0.00	0.00	**32.83**	0.00
Frostbite	269.87	304.23	**1266.73**	312.13
Gopher	3923.13	**99855.53**	6451.07	29603.60
Gravitar	377.33	387.00	**1042.67**	987.50
IceHockey	−6.39	−3.97	−5.11	**−3.50**
Jamesbond	453.83	457.50	**683.67**	475.00
Kangaroo	507.33	1524.67	11583.67	**13360.67**
Krull	8935.40	**9115.73**	8718.40	7812.03
KungFuMaster	25395.00	30002.33	34292.00	**46761.67**
MontezumaRevenge	0.00	0.00	0.00	**805.33**
MsPacman	2220.63	4892.33	3502.20	**7516.21**
NameThisGame	5977.63	**15640.83**	6011.03	9232.70
Pitfall	−65.50	−7.64	**−1.94**	−7.40
Pong	20.21	20.80	20.69	**20.95**
PrivateEye	49.24	**99.00**	97.33	92.61
Qbert	16289.08	22051.67	21830.17	**27619.33**
Riverraid	9680.33	**17794.03**	11841.03	13966.67
RoadRunner	35918.33	40428.67	50663.33	**58728.00**
Robotank	4.30	4.89	18.54	**33.69**
Seaquest	1485.33	1739.87	**1953.53**	1851.87
SpaceInvaders	1894.02	**3140.17**	2124.57	2699.33
StarGunner	55469.33	**65005.00**	63375.67	63905.00
Tennis	−22.22	−11.26	−6.72	**−5.27**
TimePilot	3359.00	7012.00	7535.67	**10789.00**
Tutankham	105.28	**291.09**	206.42	268.24
UpNDown	30932.20	159642.17	173208.13	**279107.53**
Venture	0.00	0.00	**0.00**	0.00
VideoPinball	21061.76	373803.36	220680.47	**377935.99**
WizardOfWor	1256.33	2973.00	5744.67	**10637.33**
Zaxxon	17.00	89.33	8872.67	**16801.33**