Proceedings of Machine Learning Research 101:395–410, 2019 ACML 2019

Unified Policy Optimization for Robust Reinforcement Learning

Zichuan Lin LINZC16@MAILS.TSINGHUA.EDU.CN
Tsinghua University, Beijing

Li Zhao LIZO@MICROSOFT.COM
Microsoft Research, Beijing

Jiang Bian JIANG.BIAN@MICROSOFT.COM
Microsoft Research, Beijing

Tao Qin TAOQIN@MICROSOFT.COM
Microsoft Research, Beijing

Guangwen Yang YGW@TSINGHUA.EDU.CN
Tsinghua University, Beijing

Abstract

Recent years have witnessed significant progress in solving challenging problems across various domains using deep reinforcement learning (RL). Despite the success, the weak robustness has risen as a big obstacle for applying existing RL algorithms into real problems. In this paper, we propose unified policy optimization (UPO), a sample-efficient shared policy framework that allows a policy to update itself by considering different gradients generated by different policy gradient (PG) methods. Specifically, we propose two algorithms called UPO-MAB and UPO-ES, to leverage these different gradients by adopting the idea of multi-arm bandit (MAB) and evolution strategies (ES), with the purpose of finding the gradient direction leading to more performance gain with less extra data cost. Extensive experiments show that our approach can lead to stronger robustness and better performance than baselines. [1]

Keywords: unified policy optimization, robustness, multi-arm bandit, evolution strategies

1. Introduction

Deep reinforcement learning (RL) methods have shown their tremendous success in learning complex skills for agents and solving challenging control tasks in high-dimensional raw sensory state-space (Mnih et al., 2015; Schulman et al., 2015). The great potential of deep RL, unfortunately, cannot conceal a big concern in terms of its weak robustness, which has been a major hurdle for applying deep RL algorithms into real-world problems. In particular, such weak robustness is usually specified by inconsistent behaviors of existing RL algorithms given different hyper-parameters, environments, and random initializations (Henderson et al., 2017).

Despite the importance of robustness, there have been very limited works that attempt to increase the robustness by considering: 1) noise (Pattanaik et al., 2018; Tretschk et al., 2018; Behzadan and Munir, 2018; Havens et al., 2018); 2) step-size (Schulman et al., 2015; Wu et al., 2017; Schulman et al., 2017); 3) random initialization (Liu et al., 2017); 4) hyper-parameter (Laroche and Feraud, 2017). Meanwhile, none of them paid attention to an important aspect of the weak robustness across

1. Video is available at `https://sites.google.com/view/upopaper`

different tasks. For example, ACKTR (Wu et al., 2017) can learn a good policy in the HalfCheetah environment, while struggling in the Swimmer environment; TRPO (Schulman et al., 2015) can perform well in the Swimmer environment, while getting trouble in the HalfCheetah environment. In this paper, we propose to enhance the robustness of deep RL across different tasks by combining existing popular policy gradient (PG) algorithms. Intuitively, combining the advantages of each PG algorithms in a proper way may lead to robust performance.

A straightforward way to achieve this is to run different PG algorithms to learn different policies and combine these policies via ensemble methods (Dietterichl, 2002; Zhou, 2012). However, due to the on-policy nature of most PG methods, such as TRPO, PPO (Schulman et al., 2017), and ACKTR, each policy can only learn from experiences that generated by itself. As a result, the simple ensemble approach will lead to the linear increment of training sample cost with the number of policies, which is quite significant and unaffordable for learning complex real-world problems.

To address this problem, we propose unified policy optimization(UPO), a sample-efficient shared policy framework for combining a variety of PG methods. In this framework, there is a shared policy which is used to generate experiences for training. From the basis of such shared experiences, different PG algorithms will produce their own gradients, which will be further unified to generate a gradient to update the shared policy.

To combine a variety of gradients from different PG algorithms, a general idea is to find the gradient direction that can lead to the best performance gain. To estimate the performance gain of different gradient direction, we can leverage either extra data (sampled using new policy) or history data, where, however, extra data may introduce extra cost and historical data may not be accurate enough for estimation. To trade-off between the amount of extra data consumption and the estimation accuracy of right gradient direction, we propose two algorithms, called UPO-MAB and UPO-ES. Particularly, UPO-MAB uses historical information to estimate which gradient direction is better, with no extra data cost, and UPO-ES uses extra data to test how good each gradient direction is at each time step. Experiments demonstrate that UPO-ES works better and can beat baseline PG methods on all environments thus achieving strong robustness across different tasks. We also show that both UPO-MAB and UPO-ES outperforms the naive ensemble method.

Given the recent concerns in reproducibility (Henderson et al., 2017), we run all experiments across a large number of seeds with fair evaluation metrics to guarantee the reproducibility of our results.

To summarize, our contributions are as follows:

1. To our best knowledge, we are the first to address the robustness across different tasks.
2. We highlight two technique contributions in this paper: 1) we propose a sample-efficient shared policy framework called UPO; 2) we propose two algorithms (UPO-MAB and UPO-ES) to trade-off between extra data cost and estimation accuracy of performance gain of gradient direction.
3. We conduct a lot of experiments and in-depth analysis. Our algorithms outperform both baseline PG methods and the naive ensemble method.

2. Background

RL aims at learning the policy for agents, facing a sequential decision making problem, by interacting with environment. And, such interactions with the environment take place at discrete time steps

$(t = 0, 1, ...)$. We denote environment state space by $\mathcal{S}$, the action space $\mathcal{A}$ and the reward space $\mathcal{R}$. At time t the agent observes state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$, which results in a scalar reward $r_t \in \mathcal{R}$ and a transition to the next state $s_{t+1} \in \mathcal{S}$. We consider finite horizon problems with a discounted return objective $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $\gamma \in (0, 1]$ is the discount factor. The goal of the agent is to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes its expected discounted return.

2.1. Policy Gradient Methods

Policy Gradient (PG) is one of the most promising branches in reinforcement learning, due to its recent success in solving complex continuous motion control problems. In particular, PG methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The most commonly used gradient estimator has the form $\hat{g} = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t]$, where π_θ is a policy network represented by parameter θ and $\hat{A}_t$ is an advantage function that estimates the future discounted return at time step t; the expectation $\mathbb{E}_{\pi_\theta}[\cdots]$ indicates the empirical average over a finite batch of samples.

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015): This algorithm maximizes a certain surrogate objective function to guarantee policy improvement with non-trivial step sizes. Specifically, the objective function is maximized subject to a constraint on the size of the policy update:

$$\underset{\theta}{\text{maximize}} \ \mathbb{E}_{\pi_\theta}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \tag{1}$$

$$\text{subject to } \mathbb{E}_{\pi_\theta}[\text{KL}(\pi_\theta(\cdot|s_t), \pi_{\theta_{old}}(\cdot|s_t))] \leq \delta. \tag{2}$$

where θ_{old} is the vector of policy parameters before the update. After making a linear and quadratic approximation to the objective and constraint, respectively, this problem can be approximately solved in a more efficient way.

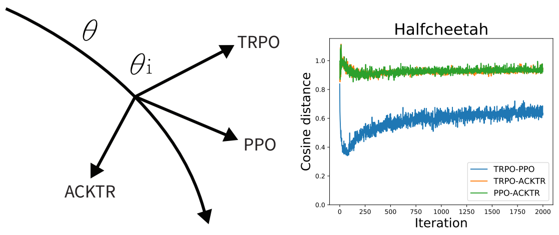
Proximal Policy Optimization (PPO) (Schulman et al., 2017): This method reformulates the constraint (2) as a clipping objective:

$$\underset{\theta}{\text{maximize}} \ \mathbb{E}_{\pi_\theta}[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3}$$

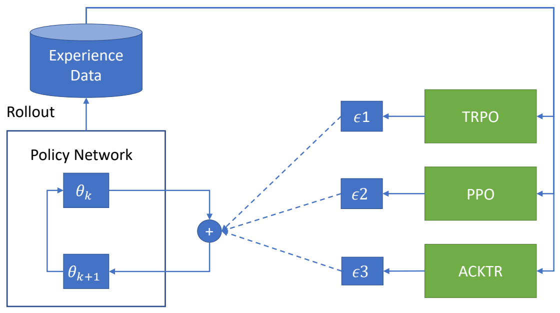
where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ and ϵ is a hyper-parameter. PPO allows the usage of multiple epochs of stochastic gradient ascent to perform each policy update. Thus, it has the same stability and reliability as trust-region methods and can result in better overall performance than TRPO.

Actor Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al., 2017): This approach uses actor-critic methods which estimate the action-value function $Q(s, a)$ and optimize a policy that maximizes discounted future return. It uses Kronecketer-factored approximation to compute the natural gradient update and applies the natural gradient update to both the actor and the critic.

3. Unified Policy Optimization



(*a*)



(*b*)

Figure 1: (a) Left: different policy gradient directions; Right: cosine distance between policy gradients that generated by three different policy optimization methods (TRPO, PPO, ACKTR).; (b) Unified Policy Optimization Framework.

To investigate the reason why PG methods are not robust across various tasks, we train a policy and generate gradients at each iteration using different PG algorithms (e.g., TRPO, PPO, ACKTR), and show the cosine distance between different policy gradients in Figure 1a. We find that gradients that generated by various methods are quite different. Gradients from ACKTR are almost orthogonal with gradients from TRPO and PPO. Gradients from TRPO and gradients from PPO are relatively close due to the similarity of their design principle.

Although gradient directions generated by various PG methods are quite different, these methods can still achieve good performance by training using their own gradients, as shown in Figure 3. Therefore, we conjecture that we can combine different policy gradients to achieve better exploration in gradient space and find a better policy optimization path.

To increase the robustness of PG algorithms across various tasks, we propose a unified policy optimization (UPO) framework, which can address the sample efficiency issue for combining various on-policy methods by sharing a unified policy network across various algorithms. After that, we will present two specific algorithms for UPO, i.e., UPO-MAB and UPO-ES.

3.1. Framework for Unified Policy Optimization

Sharing training data is common in supervised learning due to static property of the dataset. However, this is not the case in RL. The incoming data an agent can receive depends on agent itself. For off-policy RL algorithms, data generated by different agents can be shared since they can leverage off-policy training data naturally. But for on-policy RL algorithms, experience sharing is difficult since each policy can only learn from experiences generated by itself (Cauwet et al., 2016).

To overcome the obstacle of experience sharing between on-policy RL algorithms, we propose to share the policy. Specifically, we share policy parameter and policy architecture between different on-policy RL algorithms. Note that we do not share value functions because: 1) keeping separate value functions does not have influence on experience sharing; 2) values functions from different PG methods can be trained according different logics and help to generate different policy gradients.

Figure 1b shows the unified policy optimization framework. The model consists of three components, including a shared policy network, shared experience data, and a library of different on-policy PG methods. We denote policy parameter at k-th training iteration by θ_k. At k-th training iteration, we first roll out B steps (B is batch-size) using the shared policy π_{θ_k} to get shared training experience, and then leverage different on-policy PG methods to generate different policy gradients $(\epsilon_1, \epsilon_2, ..., \epsilon_n)$, where n is the number of PG algorithms used in UPO. Specifically, each PG method computes its own loss $J_i(\theta)$ and then generates the corresponding gradient, as shown below:

$$\epsilon_i = \nabla_\theta J_i(\theta), \forall i \in \{1, 2, ..., n\}. \tag{4}$$

Our general idea is to update the policy, in the learning iterations, into a better gradient direction. Therefore, it is essential to estimate the performance gain of each gradient and then generate a final gradient by a combination of all weighted by their respective performance:

$$\theta_{k+1} = \theta_k + \sum_{i=1}^{n} w_i \epsilon_i. \tag{5}$$

3.2. Algorithms for Unified Policy Optimization

Given the framework, we can generate different gradients by different PG algorithms at each iteration. Then, we need to decide how to combine these gradients to generate a better gradient direction. Intuitively, the requisite is to determine which gradient direction can lead to more significant performance improvement.

To compute the performance gain of a gradient direction, we can leverage the measurement by $F(\theta+\epsilon)-F(\theta)$, where $F(\theta)$ represents performance of policy π_θ. The next question is on which data to compute such measurement. In fact, if we want to estimate good gradient direction accurately, it's desirable to take extra data (interaction with environment) to get feedback. Henceforth, there exists a trade-off between the amount of extra data consumption and estimation accuracy of right gradient direction. More concretely, using too much extra data will hurt sample efficiency; on the other hand, no extra data may result in less accurate estimation of the performance gain of each gradient, which will hurt performance at last. To address such trade-off, we propose UPO-MAB and UPO-ES. In particular, UPO-MAB uses historical data to estimate the performance of each gradient direction. While it consumes no extra data, it may not make the very accurate estimation. On the contrary, UPO-ES uses extra data to judge each gradient direction. At the expense of its cost in using extra data, UPO-ES can make relatively more accurate estimation.

3.2.1. UPO-MAB

The first method we propose is called UPO-MAB. It estimates the performance gain of each gradient direction without taking extra data. Here we formulate the selection process as a multi-armed bandit

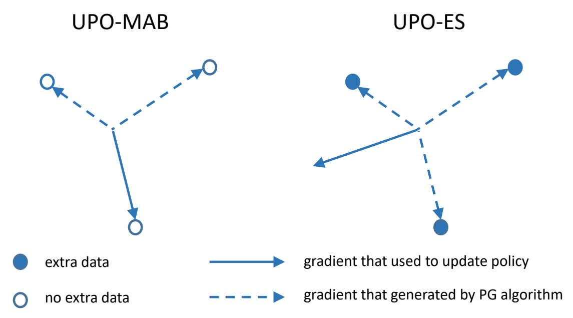

UPO-MAB
UPO-ES
extra data
no extra data
gradient that used to update policy
gradient that generated by PG algorithm

Figure 2: Illustration of UPO-MAB and UPO-ES. 'extra data' means UPO-ES needs data sampled by using new policy $\pi_{\theta+\epsilon}$ to estimate $F(\theta+\epsilon)$, while 'no extra data' means UPO-MAB does not need data sampled by using new policy.

(MAB) problem, which leverages history statistics to make selection. The underlying assumption is that, if an algorithm is good in the historical update, then it could be also good in current update. As shown in Figure 2, UPO-MAB takes no extra data to select one gradient direction to update the shared policy.

We define the reward in this MAB problem as the performance improvement $F(\theta+\epsilon_i) - F(\theta)$ of selecting the gradient generated by algorithm i. We adopt UCB1 algorithm to solve this bandit problem, which is popularly used in previous work (Li and Zhang, 2017; Laroche and Feraud, 2017). Specifically, we use $Q(i)$ to estimate how good is taking gradient by algorithm i. In general, $Q(i)$ is a statistic that represents on average how much performance gain can we get by taking algorithm i. As a result, we do not need to take extra data at each iteration, but to select the algorithm with best $Q(i)$. In order to balance between exploration and exploitation, UCB1 also adds a confidence bound to consider variance and bias of each action. Since the reward in MAB is assumed to be in range $[0, 1]$, we further normalize the reward and apply sigmoid function to satisfy this assumption.

Since reward distribution of each bandit is changing during the training process, the algorithm selection here is a stochastic MAB problem with non-stationary rewards (Besbes et al., 2014). Previous works consider forgetting the past (Garivier and Moulines, 2011). Here we use a forgetting factor α and update the bandit reward as $Q(i) = (1-\alpha)Q(i) + \alpha R_i$, where the hyper-parameter α controls the 'forget' degree about history. Algorithm 1 shows the pseudo code of our approach.

3.2.2. UPO-ES

On top of guessing which gradient direction is better according to the historical information, an alternative way is to test how well each gradient direction is at current training iteration. As shown in Figure 2, UPO-ES uses extra data to test on all gradient directions generated by different PG methods, and then generates a new gradient.

Specifically, we roll out T steps ($T \ll B$) for each new policy $\pi_{\theta_k+\epsilon_i} (i = 1, 2, ..., n)$ and compute the performance improvement. Here testing data will introduce extra sample cost. We measure the performance of policy π_θ by $F_T(\theta)$, which represents the T-step Monte-Carlo return. Formally, the testing function is defined as $F_T(\theta) = \sum_{t=1}^{T} r_t$. Then the performance improvement of i-th gradient is defined by $R_i = F_T(\theta_k + \epsilon_i) - F_T(\theta_k)$. To stabilize R_i, we firstly maintain running average and running standard deviation of the performance gain, and then we will use these values to normalize R_i for each gradient direction.

Intuitively, if the performance gain R_i of the gradient generated by algorithm i is higher, we will consider the corresponding gradient more. Furthermore, we apply a softmax operation to adjust the smoothness of R_i:

$$w_i = \frac{\exp(R_i/\tau)}{\sum_{j=1}^{n}\exp(R_j/\tau)}, i = 1, 2, ..., n, \tag{6}$$

where τ is the temperature that decides the smoothness of softmax operator. This allows a soft combination of different gradients. There are two special cases in this method: when $\tau \to 0$, our approach will select a gradient direction that achieves the best result in testing time at each training iteration; when $\tau \to \infty$, our approach approximates to average all gradients. This method is similar to Evolution Strategies (ES) (Salimans et al., 2017). ES updates policy by evolving among sampling Gaussian noise, while our method updates the shared policy by evolving among different gradients. Algorithm 2 shows the pseudo code of our approach.

Although UPO-ES takes extra data to evaluate performance gain, we empirically find that a small amount of testing data is enough to achieve good performance. This suggests that extra data will not hurt sample efficiency too much.

Algorithm 1 UPO-MAB

```
Initialize: $Q(i) = 0, T(i) = 1, i = 1, ..., n$
Generate training experience with initial policy $\pi_{\theta_0}$
for $k = 0$ to $K$ do
    Generate all policy gradients $\epsilon_j, \forall j = 1, ..., n$
    Select an algorithm: $i = \text{argmax}_{j=1,...,n}(Q(j) + \sqrt{\frac{2\ln(\sum_{l=1}^{n} T(l))}{T(j)}})$
    Compute gradient: $\epsilon_i = \nabla_\theta J_i(\theta)$
    Update policy: $\theta_{k+1} = \theta_k + \epsilon_i$
    Generate next batch of training data with $\pi_{\theta_{k+1}}$
    Get performance improvement as reward: $R_i = F(\theta_{k+1}) - F(\theta_k)$
    Reward normalization: $R_i = \text{normalize}(R_i)$
    Apply sigmoid: $R_i = \frac{1}{1+exp(-R_i)}$
    $Q(i) = (1 - \alpha)Q(i) + \alpha R_i$
    $T(i) = T(i) + 1$
end
```

Algorithm 2 UPO-ES

```
Generate training experience with initial policy $\pi_{\theta_0}$
for $k = 0$ to $K$ do
    Compute gradients: $\epsilon_i = \nabla_\theta J_i(\theta), \forall i \in \{1, ..., n\}$
    Roll out $T$ steps for each policy $\pi_{\theta_k + \epsilon_i}, \forall i \in \{1, ..., n\}$
    Get performance improvement as reward: $R_i = F_T(\theta_k + \epsilon_i) - F_T(\theta_k), \forall i \in \{1, ..., n\}$
    Reward normalization: $R_i = \mathrm{normalize}(R_i), \forall i \in \{1, ..., n\}$
    Compute weights $w_i$ using Eq. 6
    Update policy: $\theta_{k+1} = \theta_k + \sum_{i=1}^{n} w_i \epsilon_i$
    Generate next batch of training data with $\pi_{\theta_{k+1}}$
end
```

4. Discussion

Theoretical foundation. Our proposed algorithms, UPO-MAB and UPO-ES, theoretically can be viewed as a simplified version of meta RL algorithms (Wang et al., 2016a). Specifically, the policy gradient method selection in UPO is essentially a meta RL problem with a meta policy π^m whose objective is to maximize final performance of the policy after its training with UPO is over. In particular, the state of π^m is defined as parameters of current policy, the action of π^m is defined as the selection among multiple PG methods, and the reward of each action is defined as the performance gain of new obtained policy over the old one.

However, such meta RL learning process requires amounts of training episodes to learn the meta policy, which is quite inefficient. To derive practical algorithms, UPO-MAB and UPO-ES are proposed with certain assumptions and approximations for the purpose of simplification. Specifically, UPO-MAB is introduced with the assumption that the state (policy parameters) will not change dramatically in recent policy updates, especially when our paper does not only set a small learning rate for policy update but also uses forgetting reward update (α=0.3) to forget historical state. Since the recent states are roughly the same, we can consider that they are facing the same MAB problem. It is thus natural to adopt an MAB algorithm (UCB1) to solve this problem. Meanwhile, UPO-ES, as a greedy variant of Evolution Strategies, is proposed with the assumptions including: 1) taking action greedily with respect to current state will lead to a not bad final total reward; 2) linear combination of actions with respect to their rewards may lead to even better gradient.

On-policy or off-policy. Our UPO framework is mainly based on *on-policy* gradients, but it can also incorporate *off-policy* (Lillicrap et al., 2015; Haarnoja et al., 2018; Fujimoto et al., 2018; Lin et al., 2018) methods in principle. However, leveraging off-policy experience in UPO raises another problem about the combination of off-policy and on-policy gradients (Wang et al., 2016b; Gu et al., 2017; Oh et al., 2018). Due to the limit of space, we might leave it in future works.

5. Experiments

We conduct the robotic locomotion experiments using the MuJoCo simulator (Todorov et al., 2012). We build all network architectures following common practices (Schulman et al., 2015, 2017; Wu et al., 2017). We use three (n=3) PG algorithms (i.e., TRPO, PPO, ACKTR) for UPO framework

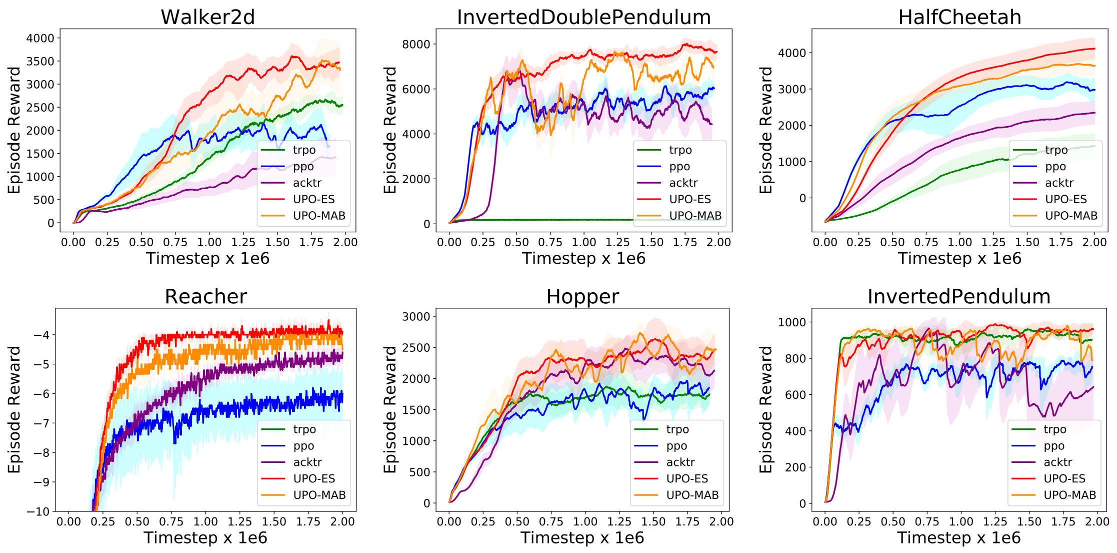


Figure 3: Performance comparisons on six standard MuJoCo environments trained for 2 million time steps. The shaded region denotes the standard deviation over 10 random seeds.

in our experiments. Our algorithms are built based on the implementation of OpenAI baselines code (Dhariwal et al., 2017). Batch-size is set to 2048 for all experiments.

We conduct hyper-parameter search in the HalfCheetah and Walker2D environments. For UPO-MAB, we tune hyper-parameter α among $\{0.01, 0.1, 0.2, 0.3, 0.4\}$. For UPO-ES, We tune τ and T in the range of $\{0.1, 1, 3, 6, 9\}$ and $\{100, 150, 200, 300\}$ respectively. Finally we set $\tau = 6$, $T = 200$ and $\alpha = 0.3$.

Each algorithm has been implemented on 7 environments. We score each run of the algorithms by computing the average total reward of the last 100 episodes. Note that the training curves of our method UPO-ES includes the extra data for fair comparison.

5.1. Comparison with Strong Baselines

Figure 3 shows the performance comparison between TRPO, PPO, ACKTR, UPO-MAB and UPO-ES. We show the scores comparison in Table 1. Similar to previous work (Schulman et al., 2017), we shift and scale the scores for each environment so that the random policy gives a score of 0 and the best result is set to 1, and average over all environments to produce a single scalar for each algorithm. To compare with naive ensemble method, we also include UPO-random and UPO-average as baselines. We find that both UPO-ES and UPO-MAB outperform baseline methods in averaged normalized performance. This shows that UPO-MAB and UPO-ES can achieve robust and consistent performance across different environments. We also find that UPO-ES performs much better than UPO-MAB. This suggests that a small amount of extra data can lead to better estimation accuracy. We show the video of trained policies at `https://sites.google.com/view/upopaper`. From the video, we observe that all baseline algorithms stuck into different bad local optima while UPO-ES can avoid such local optima and show much better behavior.

It is also worth noting that UPO-ES has a high tolerance for its subcomponents' performance variation. In the InvertPendulum environment, the methods PPO and ACKTR have very high vari-

	TRPO	PPO	ACKTR	UPO-MAB	UPO-ES	UPO-rand	UPO-avg
HalfCheetah	1438	3191	2347	3695	**4113**	3347	3167
Hopper	1865	1964	2483	**2736**	2652	2210	2409
Walker2D	2667	2137	1429	3521	**3608**	3276	3380
InvertedPendulum	960	788	966	980	**988**	965	971
InvertedDoublePendulum	189	6128	6825	7583	**8270**	7246	8113
Swimmer	112	108	62	113	**118**	107	110
Reacher	-110	-5.8	-4.5	-4.0	**-3.5**	-5.3	-4.2
Avg. Normalized Score	0.53	0.79	0.75	0.94	**0.99**	0.89	0.92

Table 1: Average score results over 10 random seeds. UPO-rand represents UPO-random, which randomly picks one algorithm to train the shared policy at each training step; UPO-avg represents UPO-average, which uses average of all gradients to update the shared policy. Higher average normalized score means better robustness.

ance during training. Although UPO-ES is built based on these methods, it is not affected by them and can still demonstrate strong robustness in the training process. In InvertDoublePendulum, the performance of TRPO does not get off the ground in the whole training process, however, both UPO-ES and UPO-MAB can still take advantages from all subcomponents and can achieve better performance. In HalfCheetah, Walker2D and Hopper environments, PPO shows the fastest learning efficiency at the beginning. However, it goes into local optimal after 0.75 million training time steps and cannot get further performance improvements in later stage of training. In this case, UPO-ES can take the advantage of early learning efficiency of PPO without falling into PPO's local optima at the later training stage.

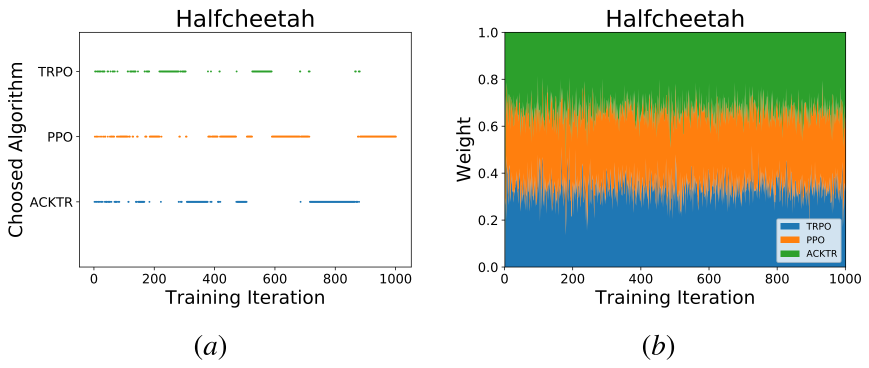


Figure 4: Figure(a) shows the algorithm selected by bandit at each training iteration. Figure(b) shows the change of weight w_i of each algorithm at each training iteration.

5.2. Gradient Choosing Behaviors of UPO-MAB and UPO-ES

To gain more constructive insights about our methods, we intend to have a closer look on how UPO-MAB and UPO-ES leverage the three gradients. We show the gradient choosing behaviors of UPO-MAB and UPO-ES in Figure 4.

Figure 4(a) shows bandit behavior of UPO-MAB during training. We find that UPO-MAB can select different algorithms in different stage of training. In the beginning, UPO-MAB conducts exploration across different PG algorithms by trying each bandit frequently. After adequate trials, it begins to leverage different PG algorithms according to the historical reward record. In HalfCheetah, we observe that UPO-MAB prefers to use PPO and ACKTR in later stage of training. This makes sense because PPO and ACKTR perform well in this environment.

Figure 4(b) shows change of weight w_i of UPO-ES in the training process. We find that the value of three weights are changing according to their performance estimation all the time. Different from UPO-MAB which chooses one gradient each time, UPO-ES performs a soft combination of these three gradients to generate a new gradient. Due to this soft combination, UPO-ES does not incline too much to any gradient, which allows it to explore better than UPO-MAB in gradient space.

5.3. Ablation Study

Ablation Study for UPO-ES In order to have a closer look at the importance of each policy gradient, we conduct an ablation study for UPO-ES. We compare four methods: 'UPO-ES', 'UPO-ES w/o TRPO', 'UPO-ES w/o PPO' and 'UPO-ES w/o ACKTR'. The last three methods represents UPO-ES without one PG algorithm. We train different policies for these four methods using the hyper-parameter $\tau = 6, T = 200$ and show the training curves in Figure 5. We find that UPO-ES demonstrates strongest robustness. In HalfCheetah, the method 'UPO-ES w/o PPO' has the largest drop on performance, which indicates the gradient from PPO is more important than the others. In Hopper, the performance of method 'UPO-ES w/o ACKTR' has large gap with that of other methods, showing that ACKTR can provide more valuable information during training in this environment.

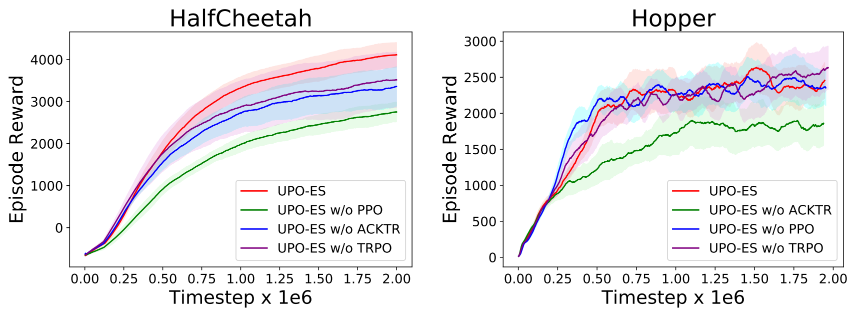


Figure 5: Ablation Study for UPO-ES.

Ablation Study for UPO-MAB We also conduct an ablation study for UPO-MAB. We compare four methods: 'UPO-MAB', 'UPO-MAB w/o TRPO', 'UPO-MAB w/o PPO' and 'UPO-MAB w/o ACKTR'. The last three methods represents UPO-MAB without one PG algorithm. The hyper-parameter α is set as 0.3. In Hopper, the performance of method 'UPO-MAB w/o TRPO' has large gap with that of other methods, showing that TRPO can provide pivotal information during training in this environment. However, in HalfCheetah, we surprisingly find that UPO-MAB does not demonstrate strongest robustness. The method 'UPO-MAB w/o ACKTR' even performs better than 'UPO-MAB'. We conjecture this is because the way that chooses one gradient each time like UPO-MAB is less robust than soft combination like UPO-ES.

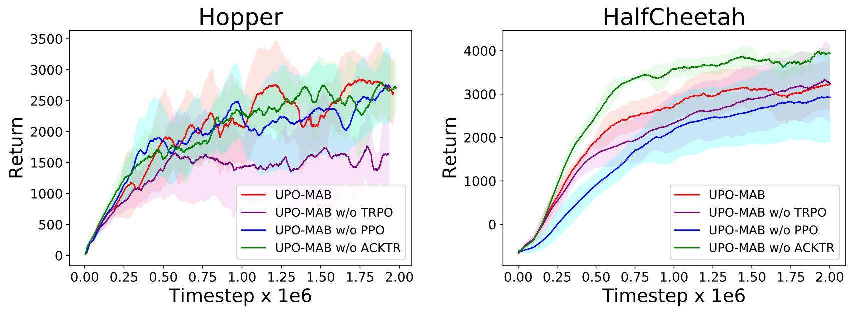


Figure 6: Ablation study for UPO-MAB.

5.4. Effects of Hyper-parameter

In this section, we show the effects of hyper-parameter. In Figure 7(left), we find that performance can be improved with more extra data when T varies between 100 and 200. This suggests that the performance estimation is more accurate with more extra data. On the other hand, too much extra data (e.g., $T = 300$) will hurt the sample efficiency. To trade off between sample efficiency and estimation accuracy, we choose $T = 200$.

Figure 7(right) also shows the effect of hyper-parameter α in UPO-MAB. We find that the performance of UPO-MAB drops when using too large or too small values. We conjecture the reason is because: when using small value, the assumption of UPO-MAB might not hold well; when using large value, the bandits' Q-values become unstable.

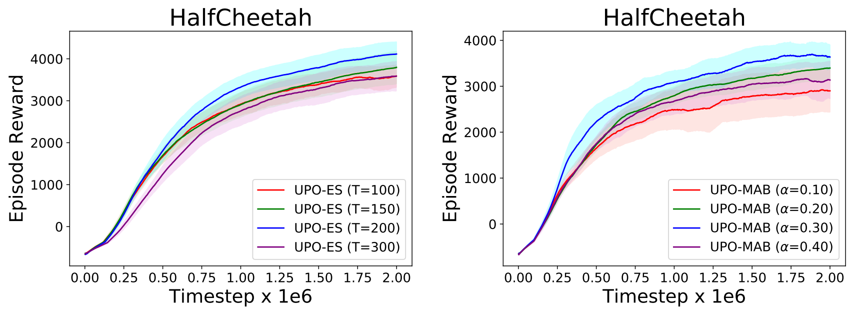


Figure 7: Effects of hyper-parameters.

5.5. Effects of Extra Data on Sample Efficiency

In order to examine the effects of extra data on sample efficiency, we compare UPO-ES and 'UPO-ES (w.o extra data)' in Figure 8. The two curves are only different in the statistics of x-axis timestep. The curve of 'UPO-ES (w.o extra data)' does not count the extra data in x-axis. We observe that the use of extra data has negligible influence on performance, which indicates the UPO-ES is able to use cheap extra data to leverage evolution strategies to improve performance.

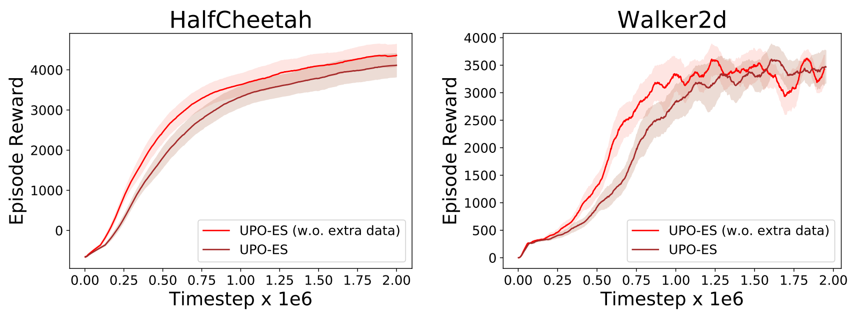


Figure 8: Effects of extra data on sample efficiency.

6. Related Work

Previous works considered addressing robustness of RL. Pattanaik et al. (2018) improve robustness by injecting adversarial attacks. Tretschk et al. (2018) add sequential attacks on agents for long-term adversarial goals. Behzadan and Munir (2018) use parameter noise to mitigate policy manipulation attacks. Havens et al. (2018) consider online robust policy learning in the presence of unknown adversaries. Schulman et al. (2015, 2017) consider address robustness of policy learning via controlling step size. Liu et al. (2017) address the robustness of policy initialization. Laroche and Feraud (2017) consider training RL algorithms with different hyper-parameters, thus training a robust policy, in which hyper-parameters are unnecessary to be tuned. None of them considered to solve the weak robustness across different tasks. In our paper, we propose UPO-MAB and UPO-ES to increase the robustness across different tasks.

UPO-ES is related to Evolutionary Reinforcement Learning (ERL) (Khadka and Tumer, 2018) which hybridizes ES with PG. The key idea of ERL is to leverage the population of ES to provide diversified data to train an agent and then inject gradient information into ES. Our method UPO-ES differs from ERL in the following aspects. Firstly, ERL uses experiences from many policies to generate one gradient while UPO-ES utilizes experiences from a shared policy to generate different gradients. Secondly, ERL can only use off-policy PG methods (e.g., DDPG) to learn from experiences while UPO-ES allows many on-policy PG methods to generate different gradients. UPO-ES is also related to Evolved Policy Gradient (EPG) (Houthooft et al., 2018). EPG uses ES to update a parameterized loss function, while UPO-ES uses ES to update policy gradients based on several prior PG methods.

UPO-MAB is related to ESBAS (Laroche and Feraud, 2017) which tackles the problem of selecting online off-policy RL algorithms. The idea is to use a bandit meta-algorithm to select one algorithm to control the next trajectory in the learning process. UPO-MAB differs from this method in several aspects. Firstly, ESBAS considers many off-policy RL agents which are allowed to share training experience naturally, while UPO-MAB uses a shared policy framework to tackle the problem of sharing experience between on-policy PG methods. Secondly, ESBAS addresses robustness across different hyper-parameter settings, while UPO-MAB addresses robustness across different tasks.

7. Conclusion

In this paper, we aim to address the robustness across different tasks for policy optimization methods. To the best of our knowledge, we are the first to address this issue. We propose a sample-efficient shared policy framework called UPO which is able to generate many gradients on a batch of shared experiences. We further propose two algorithms, UPO-MAB and UPO-ES. Experiments show that our methods achieve strong robustness across different tasks. In the future, more research can be done on how to 1) incorporate more PG methods into UPO; 2) use a model to roll out for performance estimation.

References

Vahid Behzadan and Arslan Munir. Mitigation of policy manipulation attacks on deep q-networks with parameter-space noise. *arXiv preprint arXiv:1806.02190*, 2018.

Omar Besbes, Yonatan Gur, and Assaf Zeevi. Stochastic multi-armed-bandit problem with non-stationary rewards. In *Advances in neural information processing systems*, pages 199–207, 2014.

Marie-Liesse Cauwet, Jialin Liu, Baptiste Rozière, and Olivier Teytaud. Algorithm portfolios for noisy optimization. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):143–172, 2016.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

Thomas G Dietterichl. Ensemble learning. 2002.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International Conference on Algorithmic Learning Theory*, pages 174–188. Springer, 2011.

Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3846–3855, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Aaron J Havens, Zhanhong Jiang, and Soumik Sarkar. Online robust policy learning in the presence of unknown adversaries. *arXiv preprint arXiv:1807.06064*, 2018.

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

Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pages 5405–5414, 2018.

Shauharda Khadka and Kagan Tumer. Evolutionary reinforcement learning. *arXiv preprint arXiv:1805.07917*, 2018.

Romain Laroche and Raphael Feraud. Reinforcement learning algorithm selection. *arXiv preprint arXiv:1701.08810*, 2017.

Siyuan Li and Chongjie Zhang. An optimal online method of selecting source policies for reinforcement learning. *arXiv preprint arXiv:1709.08201*, 2017.

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

Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. *arXiv preprint arXiv:1805.07603*, 2018.

Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *arXiv preprint arXiv:1806.05635*, 2018.

Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

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

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

Edgar Tretschk, Seong Joon Oh, and Mario Fritz. Sequential attacks on agents for long-term adversarial goals. *arXiv preprint arXiv:1805.12487*, 2018.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016a.

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

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.

Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.