Fast Efficient Hyperparameter Tuning for Policy Gradient Methods

Supratik Paul, Vitaly Kurin, Shimon Whiteson
Deptartment of Computer Science
University of Oxford
{supratik.paul,vitaly.kurin,shimon.whiteson}@cs.ox.ac.uk

Abstract

The performance of policy gradient methods is sensitive to hyperparameter settings that must be tuned for any new application. Widely used grid search methods for tuning hyperparameters are sample inefficient and computationally expensive. More advanced methods like Population Based Training (Jaderberg et al., 2017) that learn optimal schedules for hyperparameters instead of fixed settings can yield better results, but are also sample inefficient and computationally expensive. In this paper, we propose Hyperparameter Optimisation on the Fly (HOOF), a gradient-free algorithm that requires no more than one training run to automatically adapt the hyperparameter that affect the policy update directly through the gradient. The main idea is to use existing trajectories sampled by the policy gradient method to optimise a one-step improvement objective, yielding a sample and computationally efficient algorithm that is easy to implement. Our experimental results across multiple domains and algorithms show that using HOOF to learn these hyperparameter schedules leads to faster learning with improved performance.

1 Introduction

Policy gradient methods (Williams, 1992; Sutton et al., 1999) optimise reinforcement learning policies by performing gradient ascent on the policy parameters and have shown considerable success in environments characterised by large or continuous action spaces (Mordatch et al., 2015; Schulman et al., 2016; Rajeswaran et al., 2017). However, like other gradient-based optimisation methods, their performance can be sensitive to a number of key hyperparameters.

For example, the performance of first order policy gradient methods can depend critically on the learning rate, the choice of which in turn often depends on the task, the particular policy gradient method in use, and even the optimiser, e.g., RMSProp (Tieleman and Hinton, 2012) and ADAM (Kingma and Ba, 2014) have narrow ranges for good learning rates (Henderson et al., 2018b) which may not be known a priori. Even for second order methods like Natural Policy Gradients (NPG) (Kakade, 2001) or Trust Region Policy Optimisation (TRPO) (Schulman et al., 2015), which are more robust to the KL divergence constraint (which can be interpreted as a learning rate), significant performance gains can often be obtained by tuning this parameter (Duan et al., 2016).

Similarly, variance reduction techniques such as Generalised Advantage Estimators (GAE) (Schulman et al., 2016), which trade variance for bias in policy gradient estimates, introduce key hyperparameters (γ, λ) that can also greatly affect performance (Schulman et al., 2016; Mahmood et al., 2018).

Given such sensitivities, there is a great need for effective methods for tuning policy gradient hyperparameters. Perhaps the most popular hyperparameter optimiser is simply grid search (Schulman et al., 2015; Mnih et al., 2016; Duan et al., 2016; Igl et al., 2018; Farquhar et al., 2018). More sophisticated techniques such as Bayesian optimisation (BO) (Srinivas et al., 2010; Hutter et al.,

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2011; Snoek et al., 2012; Chen et al., 2018) have also proven effective, and new innovations such as Population Based Training (PBT) (Jaderberg et al., 2017) and meta-gradients (Xu et al., 2018) have shown considerable promise. Furthermore, a host of methods have been proposed for hyperparameter optimisation in supervised learning (see Section 4).

However, all these methods suffer from a major problem: they require performing many learning runs to identify good hyperparameters. This is particularly problematic in reinforcement learning, where it incurs not just computational costs but sample costs, as new learning runs typically require fresh interactions with the environment. This sample inefficiency is obvious in the case of grid search, BO based methods and PBT. However, even meta-gradients, which reuses samples collected by the underlying policy gradient method to train the meta-learner, requires multiple training runs. This is because the meta-learner introduces its own set of hyperparameters, e.g., meta learning rate and reference (γ, λ), all of which need tuning to achieve good performance.

Furthermore, grid search and BO based methods typically estimate only the best fixed values of the hyperparameters, which often actually need to change dynamically during learning (Jaderberg et al., 2017; François-Lavet et al., 2015). This is particularly important in reinforcement learning, where the distribution of visited states, the need for exploration, and the cost of taking suboptimal actions can all vary greatly during a single learning run.

To make hyperparameter optimisation practical for reinforcement learning methods such as policy gradients, we need radically more efficient methods that can dynamically set key hyperparameters on the fly, not just find the best fixed values, and do so within a single run, using only the data that the baseline method would have gathered anyway, without introducing new hyperparameters that need tuning. This goal may seem ambitious, but in this paper we show that it is actually entirely feasible, using a surprisingly simple method we call Hyperparameter Optimisation on the Fly (HOOF).

The main idea is as follows: At each iteration, sample trajectories using the current policy. Next, generate some candidate policies and estimate their value sample efficiently by using an *off-policy* method. Finally, update the policy greedily with respect to the estimated value of the candidates. In practice, HOOF uses the policy gradient method with different hyperparameter (e.g., the learning rate, γ, and λ) settings to generate candidate policies and then uses importance sampling (IS) to construct off-policy estimates of the value of each candidate policy.

The viability of such a simple approach is counter-intuitive since off-policy evaluation using IS tends to have high variance that grows rapidly as the behaviour and evaluation policies diverge. However, HOOF is motivated by the insight that in second order methods such as NPG and TRPO, constraints on the magnitude of the update in policy space ensure that the IS estimates remain informative. While this is not the case for first order methods, we show that adding a simple KL constraint, without any of the complications of second order methods, suffices to keep IS estimates informative and enable effective hyperparameter optimisation. We further show that the performance of HOOF is robust to the setting of this KL constraint.

HOOF is 1) sample efficient, requiring no more than one training run; 2) computationally efficient compared to sequential and parallel search methods; 3) able to learn a dynamic schedule for the hyperparameters that outperforms methods that learn fixed hyperparameter settings; and 4) simple to implement. Being gradient free, HOOF also avoids the limitations of gradient-based methods (Sutton, 1992; Luketina et al., 2016; Xu et al., 2018) for learning hyperparameters. While such methods can be more sample efficient than grid search or PBT, they can be sensitive to the choice of their own hyperparameters (see Sections 4 and 5.1) and thus require more than one training run to tune their own hyperparameters.

We evaluate HOOF across a range of simulated continuous control tasks using the Mujoco OpenAI Gym environments (Brockman et al., 2016). First, we apply HOOF to A2C (Mnih et al., 2016), and show that using it to learn the learning rate can improve performance. We also perform a benchmarking exercise where we use HOOF to learn both the learning rate and the weighting for the entropy term and compare it against a grid search across these two hyperparameters. Next, we show that using HOOF to learn optimal hyperparameter schedules for NPG can outperform TRPO. This suggests that while strictly enforcing the KL constraint enables TRPO to outperform NPG, doing so becomes unnecessary once we can properly adapt NPG's hyperparameters.

2 Background

Consider the RL task where an agent interacts with its environment and tries to maximise its expected return. At timestep t, it observes the current state s_t, takes an action a_t, receives a reward $r_t = r(s_t, a_t)$, and transitions to a new state s_{t+1} following some transition probability $\mathcal{P}$. The value function of the state s_t is $V(s_t) = \mathbb{E}_{a\sim\pi, s\sim\mathcal{P}}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$ for some discount rate $\gamma \in [0, 1)$. The undiscounted formulation of the objective is to find a policy that maximises the expected return $J(\pi) = \mathbb{E}_{a\sim\pi, s\sim\mathcal{P}, s_0\sim p(s_0)}[\sum_t r_t]$. In stochastic policy gradient algorithms, a_t is sampled from a parametrised stochastic policy $\pi(a|s)$ that maps states to actions. These methods perform an update of the form

$$\pi' = \pi + f(\psi). \tag{1}$$

Here $f(\psi)$ represents a step along the gradient direction for some objective function estimated from a batch of sampled trajectories $\{\tau_1^\pi, \tau_2^\pi, \ldots, \tau_K^\pi\}$, and ψ is the set of hyperparameters. We use π to denote both the policy as well as the parameters.

For policy gradient methods with GAE, $\psi = (\alpha, \gamma, \lambda)$, and the update takes the form:

$$f(\alpha, \gamma, \lambda) = \alpha \underbrace{\sum_t \nabla \log \pi(a_t|s_t) A_t^{GAE(\gamma,\lambda)}}_{g(\gamma,\lambda)} \tag{2}$$

where $A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + ...)$ with $A_t^{(k)} = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$. By discounting future rewards and bootstrapping off the value function, GAE reduces the variance due to rewards observed far in the future, but adds bias to the policy gradient estimate. Well chosen (γ, λ) can significantly speed up learning (Schulman et al., 2016; Henderson et al., 2018a; Mahmood et al., 2018).

In first order methods, small updates in parameter space can lead to large changes in policy space, leading to large changes in performance. Second order methods like NPG address this by restricting the change to the policy through the constraint $KL(\pi'||\pi) \leq \delta$. An approximate solution to this constrained optimisation problem leads to the update rule:

$$f(\delta, \gamma, \lambda) = \sqrt{\frac{2\delta}{g(\gamma,\lambda)^T I(\pi)^{-1} g(\gamma,\lambda)}}\, I(\pi)^{-1} g(\gamma, \lambda), \tag{3}$$

where $I(\pi)$ is the estimated Fisher information matrix (FIM).

Since the above is only an approximate solution, the $KL(\pi'||\pi)$ constraint can be violated in some iterations. Further, since δ is not adaptive, it might be too large for some iterations. TRPO addresses these issues by requiring an improvement in the surrogate $\mathcal{L}_\pi(\pi') = \mathbb{E}_{a\sim\pi, s\sim\mathcal{P}}[\frac{\pi'(a|s)}{\pi(a|s)} A^{GAE(\gamma,\lambda)}]$, as well as ensuring that the KL-divergence constraint is satisfied. It does this by performing a backtracking line search along the gradient direction. As a result, TRPO is more robust to the choice of δ (Schulman et al., 2015).

3 Hyperparameter Optimisation on the Fly

The main idea behind HOOF is to automatically adapt the hyperparameters during training by greedily maximising the value of the updated policy, i.e., starting with policy π_n at iteration n, HOOF sets

$$\begin{aligned} \psi_n &= \operatorname*{argmax}_{\psi} J(\pi_{n+1}) \\ &= \operatorname*{argmax}_{\psi} J(\pi_n + f(\psi)), \end{aligned} \tag{4}$$

Given a set of sampled trajectories, $f(\psi)$ can be computed for any ψ, and thus we can generate different candidate π_{n+1} without requiring any further samples. However, solving the optimisation problem in (4) requires evaluating $J(\pi_{n+1})$ for each such candidate. Any on-policy approach would have prohibitive sample requirements, so HOOF uses weighted importance sampling (WIS) to

Algorithm 1 HOOF

```
input  Initial policy $\pi_0$, number of policy iterations $N$, search space for $\psi$, KL constraint $\epsilon$ if using
    first order policy gradient method.
 1: for n = 0, 1, 2, 3, ..., N do
 2:     Sample trajectories $\tau_{1:K}$ using $\pi_n$.
 3:     for z = 1, 2, ... Z do
 4:         Generate candidate hyperparameter $\{\psi_z\}$ from the search space.
 5:         Compute candidate policy $\pi^z$ using $\psi_z$ in (1)
 6:         Estimate $J(\pi^z)$ using WIS (5)
 7:         Compute $KL(\pi^z||\pi_n)$ if using first order policy gradient method,
 8:     end for
 9:     Select $\psi_n$, and hence $\pi_{n+1}$, according to (7) or (4)
10: end for
```

construct an off-policy estimate of $J(\pi_{n+1})$. Given sampled trajectories $\{\tau_1^{\pi_n}, \tau_2^{\pi_n}, .., \tau_K^{\pi_n}\}$, with corresponding returns $\{R_1^{\pi_n}, R_2^{\pi_n}, ..., R_K^{\pi_n}\}$, the WIS estimate of $J(\pi_{n+1})$ is given by:

$$J(\pi_{n+1}) = \sum_{k=1}^{K} \left(\frac{w_k}{\sum_{k=1}^{K} w_k} \right) R_k^{\pi_n}, \tag{5}$$

where $w_k = \frac{P(\tau_k^{\pi_n} \sim \pi_{n+1})}{P(\tau_k^{\pi_n} \sim \pi_n)}$. Since $p(\tau|\pi) = p(s_0) \prod_{i=0}^{T} \pi(a_i|s_i) p(s_{i+1}|s_i, a_i)$, the transitions cancel out and we have:

$$w_k = \frac{\prod_{i=0}^{T} \pi_{n+1}(a_i|s_i^k)}{\prod_{i=0}^{T} \pi_n(a_i|s_i^k)}. \tag{6}$$

The success of this approach depends critically on the quality of the WIS estimates, which can suffer from high variance that grows rapidly as the distributions of π_{n+1} and π_n diverge. Fortunately, for natural gradient methods like NPG, $KL(\pi_{n+1}||\pi_n)$ is automatically approximately bounded by the update, ensuring reasonable WIS estimates when HOOF directly uses (4). In the following, we consider the more challenging case of first order methods.

3.1 First Order HOOF

Without a KL bound on the policy update, it may seem that WIS will not yield adequate estimates to solve (4). However, a key insight is that, while the estimated policy value can have high variance, the relative ordering of the policies, which HOOF solves for, has much lower variance (See Appendix E for an illustrative example). Nonetheless, HOOF could still fail if $KL(\pi_{n+1}||\pi_n)$ becomes too large, which can occur in first order methods. Hence, First Order HOOF modifies (4) by constraining $KL(\pi_{n+1}||\pi_n)$:

$$\psi_n = \underset{\psi}{\operatorname{argmax}}\, J(\pi_{n+1}) \quad \text{s.t.} \quad KL(\pi_{n+1}||\pi_n) < \epsilon. \tag{7}$$

While this yields an update that superficially resembles that of natural gradient methods, the KL constraint is applied only during the search for the optimal hyperparameter settings using WIS. The direction of the update is determined solely by a first order gradient update rule, and estimation and inversion of the FIM is not required. From a practical perspective, this constraint is enforced by computing the KL for each candidate policy based on the observed trajectories, and the candidate is rejected if this sample KL is greater than the constraint.

If learning the learning rate using HOOF, we can also use the KL constraint to dynamically adjust the search bounds: At each iteration, if none of the candidates violate the KL constraint, we increase the upper bound of the search space by a factor ν, while if a large proportion of the candidates violate the KL constraint, we reduce the upper bound by ν. This makes HOOF even more robust to the initial setting of the search space. Note that this is entirely optional, and is simply a means to reduce the number of number of candidates that would otherwise need to be generated and evaluated to ensure that a good solution of (4) is found.

3.2 (γ, λ) Conditioned Value Function

If we use HOOF to learn (γ, λ), g_n has to be computed for each setting of (γ, λ). With neural net value functions, we modify our value function such that its inputs are (s, γ, λ), similar to Universal Value Function Approximators (Schaul et al., 2015). Thus we learn a (γ, λ)-conditioned value function that can make value predictions for any candidate (γ, λ) at the cost of a single forward pass. In Appendix D we present some experimental results to show that learning a (γ, λ)-conditioned value function is key to the success of HOOF.

3.3 Robustness to HOOF Hyperparameters and Computational Costs

HOOF introduces two types of hyperparameters of its own: the search spaces for the various hyperparameters it tunes, and the number of candidate policies generated for evaluation. Since the candidate policies are generated using random search, these hyperparameters express a straight up trade-off between performance and computational cost: A larger search space and larger number of candidates should lead to better solution for (4), but incur higher computational cost. However, just like in random search, the generation and evaluation of the candidate policies can be performed in parallel to reduce wall clock time. Alternatively, Bayesian Optimisation could be used to solve (4) efficiently. Finally, we note that HOOF with random search is always more computationally efficient than grid/random search over the hyperparameters with the same number of candidates, as HOOF saves on the additional computational cost of sampling trajectories for each candidate incurred by grid/random search. HOOF additionally introduces the KL constraint hyperparameter for first order methods. We show experimentally that the performance of HOOF is robust to a wide range of settings for this.

3.4 Choice of Optimiser

Throughout this paper we use random search as the optimiser for (4) to show that the simplest methods suffice. However, any gradient-free optimiser could be used instead. For example, grid search, CMA-ES (Hansen and Ostermeier, 2001), or Bayesian Optimisation (Brochu et al., 2010) are all viable alternatives.

Gradient based methods are not viable for two reasons. First, they require that $J(\pi_{n+1})$ be differentiable w.r.t. the hyperparameters, which might be difficult or impossible to compute, e.g. with the TRPO update. Second, they introduce learning rate and initialisation hyperparameters, which require tuning at the expense of sample efficiency.

4 Related Work

Most hyperparameter search methods can be broadly classified into sequential search, parallel search, and gradient based methods.

Sequential search methods perform a training run with some candidate hyperparameters, and use the results to inform the choice of the next set of hyperparameters for evaluation. BO is a sample efficient global optimisation framework that models performance as a function of the hyperparameters, and is especially suited for sequential search as each training run is expensive. After each training run BO uses the observed performance to update the model in a Bayesian way, which then informs the choice of the next set of hyperparameters for evaluation. Several modifications have been suggested to further reduce the number of evaluations required: input warping (Snoek et al., 2014) to address nonstationary fitness landscapes; freeze-thaw BO (Swersky et al., 2014) to decide whether a new training run should be started and the current one discontinued based on interim performance; transferring knowledge about hyperparameters across similar tasks (Swersky et al., 2013); and modelling training time as a function of dataset size (Klein et al., 2016). To further speed up the wall clock time, some BO based methods use a hybrid mode wherein batches of hyperparameter settings are evaluated in parallel (Contal et al., 2013; Desautels et al., 2014; Shah and Ghahramani, 2015; Wang et al., 2016; Kandasamy et al., 2018).

By contrast, parallel search methods like grid search and random search run multiple training runs with different hyperparameter settings in parallel to reduce wall clock time, but require more parallel computational resources. These methods are easy to implement, and have been shown to perform well (Bergstra et al., 2011; Bergstra and Bengio, 2012).

Both sequential and parallel search suffer from two key disadvantages. First, they require performing multiple training runs to identify good hyperparameters. Not only is this computationally inefficient, but when applied to RL, also sample inefficient as each run requires fresh interactions with the environment. Second, these methods learn fixed values for the hyperparameters that are used throughout training instead of a schedule, which can lead to suboptimal performance (Luketina et al., 2016; Jaderberg et al., 2017; Xu et al., 2018).

PBT (Jaderberg et al., 2017) is a hybrid of random and sequential search, with the added benefit of adapting hyperparameters during training. It starts by training a population of hyperparameters which are then updated periodically to further explore promising hyperparameter settings. However, by requiring multiple training runs, it inherits the sample inefficiency of random search.

HOOF is much more sample efficient because it requires no more interactions with the environment than those gathered by the underlying policy gradient method for one training run. Consequently, it is also far more computationally efficient. However, while HOOF can only optimise hyperparameters that directly affect the policy update, these methods can tune other hyperparameters, e.g., policy architecture. Combining these complementary strengths in an interesting topic for future work.

Gradient based methods (Sutton, 1992; Bengio, 2000; Luketina et al., 2016; Pedregosa, 2016; Xu et al., 2018) adapt the hyperparameters by performing gradient descent on the policy gradient update function with respect to the hyperparameters. This raises the fundamental problem that the update function needs to be differentiable. For example, the update function for TRPO uses conjugate gradient to approximate $I(\pi)^{-1}g$, performs a backtracking line search to enforce the KL constraint, and introduces a surrogate improvement constraint, which introduce discontinuities in the update and makes it non-differentiable.

A second major disadvantage of these methods is that they introduce their own set of hyperparameters, which can make them sample inefficient if they require tuning. For example, the meta-gradient estimates can have high variance, which in turn significantly affects performance. To address this, the objective function of meta-gradients introduces reference (γ', λ') hyperparameters to trade off bias and variance. As a result, its performance can be sensitive to these, as the experimental results of Xu et al. (2018) show. Furthermore, gradient based methods tend to be highly sensitive to the setting of the learning rate, and these methods introduce their own learning rate hyperparameter for the meta learner which requires tuning, as we show in our experiments. As a gradient-free method, HOOF does not require a differentiable objective and, while it introduces a few hyperparameters of its own, these do not affect sample efficiency, as mentioned in Section 3.3.

Other work on non-gradient based methods includes that of Kearns and Singh (2000), who derive a theoretical schedule for the TD(λ) hyperparameter that they show is better than any fixed value. Downey et al. (2010) learn a schedule for TD(λ) using a Bayesian approach. White and White (2016) greedily adapt the TD(λ) hyperparameter as a function of state. Unlike HOOF, these methods can only be applied to TD(λ) and, in the case of Kearns and Singh (2000), are not compatible with function approximation.

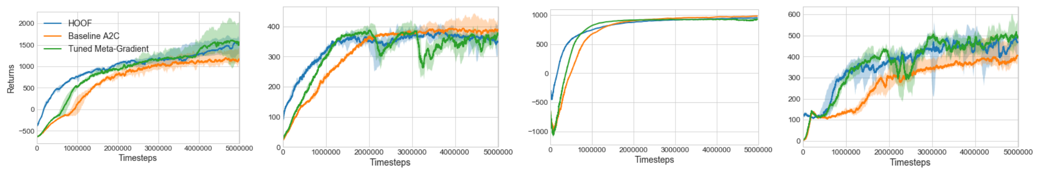
5 Experiments

To experimentally validate HOOF, we apply it to four simulated continuous control tasks from MuJoCo OpenAI Gym (Brockman et al., 2016): HalfCheetah, Hopper, Ant, and Walker. We start with A2C, and show that HOOF performs better than multiple baselines, and is also far more sample efficient. Next, we use NPG as the underlying policy gradient method and apply HOOF to learn $(\delta, \gamma, \lambda)$ and show that it outperforms TRPO.

We repeat all experiments across 10 random starts. In all figures solid lines represent the median, and shaded regions the quartiles. Similarly all results in tables represent the median. Hyperparameters that are not tuned are held constant across HOOF and baselines to ensure comparability. Details about all hyperparameters can be found in the appendices, and code is available at `https://github.com/supratikp/HOOF`.

5.1 HOOF with A2C

In the A2C framework, a neural net with parameters θ is commonly used to represent both the policy and the value function, usually with some shared layers. The update function (1) for A2C is a linear



(a) HalfCheetah (b) Hopper (c) Ant (d) Walker

Figure 1: Performance of HOOF with $\epsilon = 0.03$ compared to Baseline A2C and Tuned Meta-Gradients. The hyperparameters (α_0, β) of meta gradients had to be tuned using grid search which required 36x the samples used by HOOF.

Table 1: Performance of HOOF with different values of the KL constraint (ϵ parameter). The results show that the performance is relatively robust to the setting of ϵ.

KL constraint	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$	$\epsilon = 0.06$	$\epsilon = 0.07$
HalfCheetah	1,203	1,451	1,524	1,325	1,388	1301	1504
Hopper	359	358	350	362	359	370	365
Ant	916	942	952	957	971	963	969
Walker	466	415	467	475	456	402	457

combination of the gradients of the policy loss, the value loss, and the policy entropy:

$$f_\theta(\alpha) = \alpha\{\nabla_\theta \log \pi_\theta(a|s)(R - V_\theta(s)) + c_1 \nabla_\theta (R - V_\theta(s))^2 + c_2 \nabla_\theta H(\pi_\theta(s))\}, \tag{8}$$

where we have omitted the dependence on the timestep and other hyperparameters for ease of notation. The performance of A2C is particularly sensitive to the choice of the learning rate α (Henderson et al., 2018b), which requires careful tuning.

We learn α using HOOF with the KL constraint $\epsilon = 0.03$ ('HOOF'). We compare this against two baselines: (1) Baseline A2C, i.e., A2C with the initial learning rate set to the OpenAI Baselines default (0.0007), and (2) learning rate being learnt by meta-gradients ('Tuned Meta-Gradient'), where the hyperparameters introduced by meta-gradients were tuned using grid search.

The learning curves in Figure 1 shows that across all environments HOOF learns faster than Baseline A2C, and also outperforms it in HalfCheetah and Walker, demonstrating that learning the learning rate online can yield significant gains.

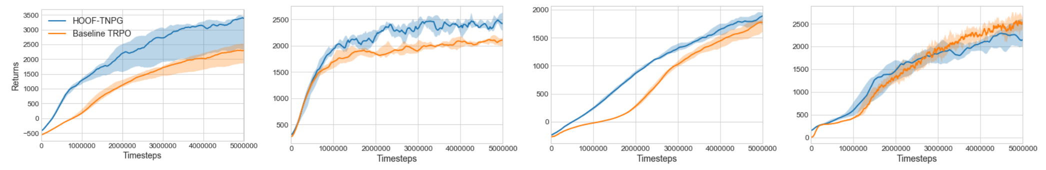
The update rule for meta-gradients when learning α reduces to $\alpha' = \alpha + \beta \nabla_{\theta'} \log \pi_{\theta'}(a|s)(R - V_{\theta'}(s))\frac{f_\theta(\psi)}{\alpha}$, where β is the meta learning rate. This leads to two issues: what should the learning rate be initialised to (α_0), and what should the meta learning rate be set to? Like all gradient based methods, the performance of meta gradients can be sensitive to the choices of these two hyperparamters. When we set α_0 to the OpenAI baselines default setting and β to 0.001 as per Xu et al. (2018), A2C fails to learn at all. Thus, we had to run a grid search over (α_0, β) to find the optimal settings across these hyperparameters. In Figure 1 we plot the best run from this grid search. Despite using 36 times as many samples (due to the grid search), meta-gradients still cannot outperform HOOF, and learns slower in 3 of the 4 tasks. The returns for each of the 36 points on the grid are presented in Appendix B.1 and they show that the performance of meta gradients can be sensitive to these two hyperparamters.

To show that HOOF's performance is robust to ϵ, its own hyperparameter quantifying the KL constraint, we repeated our experiments with different values of ϵ. The results presented in Table 1 show that HOOF's performance is stable across different values of this parameter. This is not surprising – the sole purpose of the constraint is to ensure that the WIS estimates remain viable.

Finally, to ascertain the sample efficiency of HOOF relative to grid search, we perform a benchmarking exercise. We used HOOF to learn both the learning rate and the entropy coefficient (c_2 in (8)). We split the search bounds for these across a grid with 11x11 points and ran A2C for each setting on the grid. For computational reasons we set the budget for each training run to 1 million timesteps. Given a budget of n training runs, we randomly subsample n points from the grid (without replacement) and note the best return. We repeat this 1000 times to get an estimate of the expected best return

Table 2: Comparison of sample efficiency of HOOF over grid search.

	HOOF	Max return over subsampled grid of size			
	Returns	1	2	5	10
HalfCheetah	702	-558	-241	113	354
Hopper	321	109	165	240	287
Ant	675	-7561	-272	177	476
Walker	175	99	153	224	279



(a) HalfCheetah (b) Hopper (c) Ant (d) Walker

Figure 2: Performance of HOOF-TNPG vs TRPO baselines.

of the grid search with a budget of n training runs. The results presented in Table 2 compares the returns of HOOF to that of the expected best return for grid search with different training budgets. The performance of grid search is much worse than that of HOOF with the same budget (i.e., only 1 training run). The results show that grid search can take more than 10 times as many samples to match HOOF's performance.

Appendix A.3 contains further experimental details, including results confirming that the KL constraint is crucial to ensuring sound WIS estimates.

In Appendix A.4 we show that HOOF is also robust to the choice of the optimiser by running the experiments with SGD (instead of RMSProp) as the optimiser. In this case the difference in performance is highly significant with Baseline A2C failing to learn at all.

5.2 HOOF with Truncated Natural Policy Gradients (TNPG)

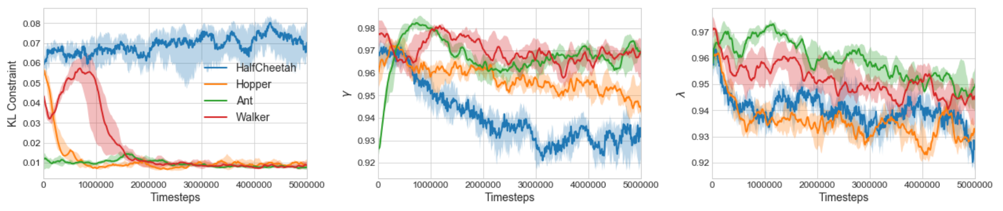
A major disadvantage of natural policy gradient methods is that they require the inversion of the FIM in (3), which can be prohibitively expensive for large neural net policies with thousands of parameters. TNPG (Duan et al., 2016) and TRPO address this by using the conjugate gradient algorithm to efficiently approximate $I(\pi)^{-1}g$. TRPO has been shown to perform better than TNPG in continuous control tasks (Schulman et al., 2015), a result attributed to stricter enforcement of the KL constraint.

However, in this section, we show that stricter enforcement of the KL constraint becomes unnecessary once we properly adapt TNPG's learning rate. To do so, we apply HOOF to learn $(\delta, \gamma, \lambda)$ of TNPG ('HOOF-TNPG'), and compare it with TRPO with the OpenAI Baselines default settings of $(\epsilon = 0.01, \gamma = 0.99, \lambda = 0.98)$ ('Baseline TRPO').

Figure 2 shows the learning curves of HOOF-TNPG and the Baseline TRPO. HOOF-TNPG learns much faster, and outperforms Baseline TRPO in all environments except for Walker where there's no significant difference. Figure 3 presents the learnt $(\delta, \gamma, \lambda)$. The results show that different KL constraints and GAE hyperparameters are needed for different domains. We could not compare with meta-gradients as the objective function is not differentiable, as discussed earlier in Section 4. We also could not perform a comparison against grid search similar to the one in Section 5.1 as the computational burden of performing a grid search over three hyperparameters was too large.

6 Conclusions & Future Work

The performance of a policy gradient method is highly dependent on its hyperparameters. However, methods typically used to tune these hyperparameters are highly sample inefficient, computationally expensive, and learn only a fixed setting of the hyperparameters. In this paper we presented HOOF, a sample efficient method that automatically learns a schedule for the learning rate and GAE hyperparameters of policy gradient methods without requiring multiple training runs. We believe that this,



(a) Learnt δ (b) Learnt γ (c) Learnt λ

Figure 3: Hyperparameters learnt by HOOF-TNPG for HalfCheetah, Hopper, and Walker.

combined with its simplicity and ease of implementation, makes HOOF a compelling method for optimising policy gradient hyperparameters.

While we have presented HOOF as a method to learn the hyperparameters of a policy gradient algorithm, the underlying principles are far more general. For example, one could compute a distribution for the gradient and generate candidate policies by sampling from that distribution, instead of just using the point estimate of the gradient. It has also been hypothesised that state/action dependent discount factors might help speed up learning (White, 2017; Fedus et al., 2019). This could be achieved by using HOOF to learn the parameters of a function that maps the states/actions to the discount factors.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement #637713), and Samsung R&D Institute UK. The experiments were made possible by a generous equipment grant from NVIDIA.

References

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*.

Brochu, E., Cora, V. M., and de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv.org.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. (2018). Bayesian optimization in alphago. *CoRR*, abs/1812.06855.

Contal, E., Buffoni, D., Robicquet, A., and Vayatis, N. (2013). Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Desautels, T., Krause, A., and Burdick, J. W. (2014). Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. `https://github.com/openai/baselines`.

Downey, C., Sanner, S., et al. (2010). Temporal difference bayesian model averaging: A bayesian perspective on adapting lambda. In *International Conference on Machine Learning*.

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

Farquhar, G., Rocktaschel, T., Igl, M., and Whiteson, S. (2018). Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. In *International Conference on Learning Representations*.

Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., and Larochelle, H. (2019). Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*.

François-Lavet, V., Fonteneau, R., and Ernst, D. (2015). How to discount deep reinforcement learning: Towards new dynamic strategies. In *NIPS 2015 Workshop on Deep Reinforcement Learning*.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018a). Deep reinforcement learning that matters. In *AAAI*.

Henderson, P., Romoff, J., and Pineau, J. (2018b). Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *CoRR*, abs/1810.02525.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*.

Igl, M., Zintgraf, L. M., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Kakade, S. (2001). A natural policy gradient. In *Advances in Neural Information Processing Systems*.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. (2018). Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*.

Kearns, M. J. and Singh, S. P. (2000). Bias-variance error bounds for temporal difference updates. In *Conference on Learning Theory*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2016). Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*.

Luketina, J., Raiko, T., Berglund, M., and Greff, K. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning*.

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 *International Conference on Machine Learning*.

Mordatch, I., Lowrey, K., Andrew, G., Popovic, Z., and Todorov, E. V. (2015). Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017). Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International Conference on Machine Learning*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*.

Shah, A. and Ghahramani, Z. (2015). Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pages 3330–3338.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. (2014). Input warping for bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*.

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*.

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*.

Swersky, K., Snoek, J., and Adams, R. P. (2014). Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Wang, J., Clark, S. C., Liu, E., and Frazier, P. I. (2016). Parallel bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*.

White, M. (2017). Unifying task specification in reinforcement learning. In *International Conference on Machine Learning*.

White, M. and White, A. (2016). A greedy approach to adapting the trace parameter for temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 557–565. International Foundation for Autonomous Agents and Multiagent Systems.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Xu, Z., van Hasselt, H. P., and Silver, D. (2018). Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*.

A A2C Experimental Details

We present further details about our A2C experiments in this section.

A.1 Implementation Details

Our codebase for the A2C experiments is based on OpenAI Baselines (Dhariwal et al., 2017) implementation of A2C and uses their default hyperparameters. Experiments involving HOOF use the same hyperparameters apart from those that are learnt by HOOF. All hyperparameters are presented in Table 3.

Table 3: Hyperparameters used for A2C experiments.

Hyperparameter	Value
Number of environments (num_envs)	40
Timesteps per worker (nsteps)	5
Total environment steps	5e6
Discounting γ	0.99
Max gradient norm	0.5
Optimiser	RMSProp
– α	0.99
– ϵ	1e-5
Policy	MLP
– Number of fully connected layers	2
– Number of units per layer	64
– Activation	tanh
Default settings for Baseline A2C (and HOOF and meta gradients if not learnt)	
– Initial learning rate	7e-4
– Learning rate schedule	linear annealing
– Value function cost weight (c_1)	0.5
– Entropy cost weight (c_2)	0.01
Grid search over (α, c_2) with A2C	
– Grid settings for α	$0.01 \times 10^{-0.5\times\{0,1,..,10\}}$
– Grid settings for c_2	$0.05 \times \{0.0, 0.1, .., 1.0\}$
HOOF specific hyperparameters	
– Initial search bounds for α	[0,1e-2]
– Number of random samples for α	100
– Search bounds for c_2 (for grid search experiment)	[0,0.2]
– Number of random samples for c_2 (for grid search experiment)	50

For HOOF, α_{UB}, the upper bound of the search space for α was dynamically updated at each iteration based on the following: if no candidates violate the KL constraint, $\alpha_{UB} \leftarrow 1.25\alpha_{UB}$. If more than 80% of the candidates violate the KL constraint, $\alpha_{UB} \leftarrow \alpha_{UB}/1.25$.

A.2 Learnt Hyperparameters

The hyperparameters learnt by HOOF are presented in Figure 4.

A.3 Performance of HOOF without a KL Constraint

Figure 5 shows that without a KL constraint HOOF does not converge, which confirms that we need to constrain policy updates so that WIS estimates remain sound.

A.4 Robustness to the Choice of Optimiser

OpenAI implementation of A2C uses RMSProp as the default optimiser. To check how robust HOOF's performance is to the choice of the optimiser, we ran both Baseline A2C and HOOF with

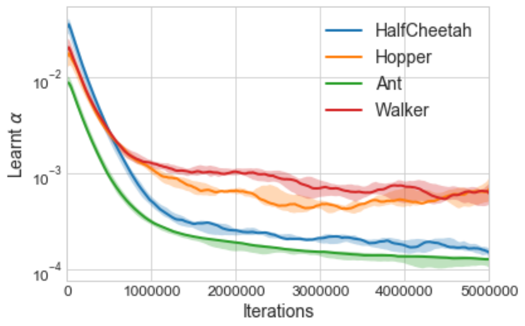
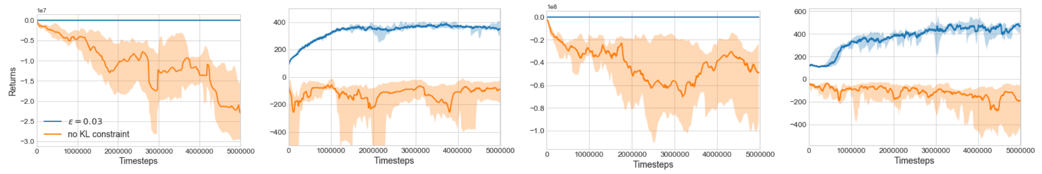


Figure 4: Schedule for the learning rates learnt by HOOF. Refer to Equation (8).



(a) HalfCheetah (b) Hopper (c) Ant (d) Walker2d

Figure 5: Comparison of the performance of HOOF with $\epsilon = 0.03$ and HOOF without any KL constraint.

SGD instead. The learning curves presented in Figure 6 shows that in this case HOOF's performance is far better than that of Baseline A2C which fails to learn at all.

B Meta-Gradients Update for α

The meta-gradient algorithm for hyperparameters ψ proceeds as follows: 1) Sample trajectories $\tau^{\theta}_{1:K} \sim \pi_\theta$. 2) Update $\theta' = \theta + f_\theta(\psi)$ (where f_θ is as per (8)). 3) Sample trajectories $\tau^{\theta'}_{1:K} \sim \pi_{\theta'}$. 4) Update $\psi' = \psi + \beta \frac{\partial J'(\tau^{\theta'}_{1:K}, \bar{\psi})}{\partial \theta'} \frac{\partial f_\theta(\psi)}{\partial \psi}$, where J' is the meta-objective with $\bar{\psi}$ the set of reference hyperparameters introduced by the meta-gradient algorithm to balance bias-variance tradeoff within the meta-objective, and β is the meta learning rate. For $\psi = \alpha$, $\frac{\partial f_\theta(\psi)}{\partial \psi} = \frac{f_\theta(\psi)}{\alpha}$, and we can use the policy loss as the meta objective, with $\frac{\partial J'(\tau^{\theta'}_{1:K}, \bar{\psi})}{\partial \theta'} = \nabla_{\theta'} \log \pi_{\theta'}(a|s)(R - V_{\theta'}(s))$.

An unconstrained meta-update can lead to α being negative. Clipping α to 0 after each meta update is not feasible since it leads to the situation where the policy does not update at all. Hence a log transform was used instead to ensure $\alpha > 0$.

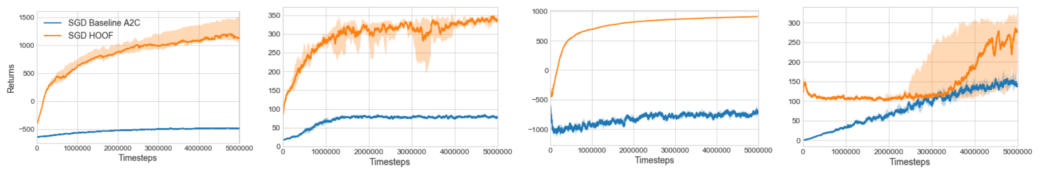
B.1 Results of Grid Search for Meta-Gradients

The returns after 5 millions timesteps for each setting of (α_0, β) on the grid is given in Table 4. Note that very few settings of the hyperparameters can match the performance of HOOF, while some settings of (α_0, β) can lead to the algorithm failing to learn at all. Setting $\alpha_0 = 1e-3$, which is closest to the OpenAI Baselines default setting, and $\beta = 1e-3$ as was used by Xu et al. (2018) in their experiments leads to performance well below that of HOOF, or even Baseline A2C.

It is also important to note that HOOF only requires 1 training run of samples (i.e. 5 million timesteps) while the grid search over the hyperparameters means that meta-gradients requires 36x samples to be able to match HOOF.

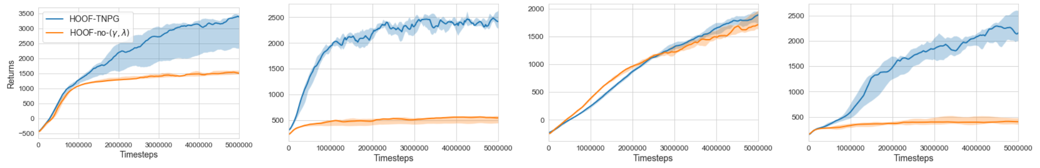
C TNPG Experimental Details

We present further details about our TNPG experiments in this section.



(a) HalfCheetah (b) Hopper (c) Ant (d) Walker2d

Figure 6: Comparison of the performance of HOOF with $\epsilon = 0.03$ and Baseline A2C where the optimiser is SGD for both (instead of RMSProp).



(a) HalfCheetah (b) Hopper (c) Ant (d) Walker

Figure 7: Performance of HOOF without (γ, λ) conditioned value functions: Not learning (γ, λ) conditioned value functions leads to significant reduction in performance in all environments except Ant.

C.1 Implementation Details

Our codebase for the TNPG experiments is based on OpenAI Baselines (Dhariwal et al., 2017) implementation of TRPO and uses their default hyperparameters. Experiments involving HOOF use the same hyperparameters apart from those that are learnt by HOOF. All hyperparameters are presented in Table 5.

D Importance of Learning (γ, λ) Conditional Value Functions

In Figure 7 we compare the performance of HOOF ('HOOF-TNPG') with that of HOOF without (γ, λ) conditional value functions ('HOOF-no-(γ, λ)'). Clearly the conditioning is key to good performance. This is because the value is highly dependent on (γ, λ) which changes throughout training.

E Illustration of the Ordering Effect of WIS

We illustrate the assertion about the relative ordering of WIS estimates through a simple experiment: Let $p(x) = N(0, 1)$ be our behaviour distribution. We are interested in $E_{q_i(x)}[X^2]$ where $q_i(x) = N(\mu_i, 1)$, $\mu_i = \{0, 1, 2, 3, 4, 5\}$. We can compute the true value analytically as $1 + \mu_i^2$. Now we compare this to a WIS estimate: we sample 10 points from $p(x)$ and use them to estimate $E_{q_i(x)}[X^2]$. We repeat this 1000 times. The boxplot of the WIS estimates in Fig 8a shows that we cannot rely on them directly as they becomes worse as $q_i(x)$ diverges from $p(x)$. However, in Fig 8b we see that the relative ordering is reliable. Note this does not guarantee that by using WIS to estimate the value of each candidate policy will always lead to a correct solution in (4). However, any factor that leads to better estimates of the policy value (for example, increasing the number of trajectories sampled) is also likely to lead to a better estimate of the relative ordering which (4) relies on.

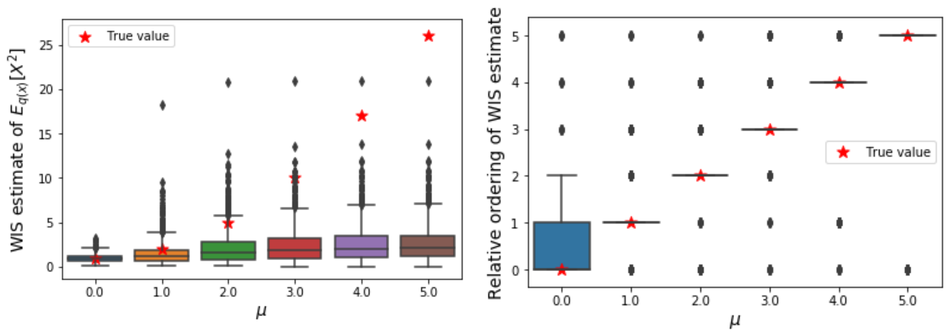
Table 4: Results of grid search over meta-gradients hyperparameters. * denotes algorithm diverged with returns $< -10^5$. Settings with returns greater than HOOF returns are shown in bold.

HalfCheetah		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-10,972	-1,230	*	*	*	-498	1523
	$1e-3$	-7,137	-221	468	1,080	**1,568**	1,272	
	$1e-4$	*	-245	313	441	-223	86	
	$1e-5$	*	-247	324	-499	-515	-518	
	$1e-6$	-641	-224	-404	-618	-616	-631	
	$1e-7$	-643	-351	-611	-633	-638	-639	

Hopper		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-2,950	-13,271	-808	-1,845	87	103	350
	$1e-3$	-12,045	-508	-801	54	**378**	**368**	
	$1e-4$	-20,086	68	67	225	215	236	
	$1e-5$	-3,309	70	65	67	61	61	
	$1e-6$	35	67	63	64	64	50	
	$1e-7$	-7,793	72	64	54	20	18	

Ant		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	*	393	*	*	*	*	952
	$1e-3$	*	761	752	950	926	884	
	$1e-4$	-47,393	-156	687	672	666	655	
	$1e-5$	*	375	588	-595	-739	-692	
	$1e-6$	*	283	373	-1,081	-1,073	-1,006	
	$1e-7$	-1,257	-514	-361	-1,017	-1,042	-964	

Walker		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-10,316	-2,922	109	294	176	159	467
	$1e-3$	-1,383	-2,636	28	445	**492**	**485**	
	$1e-4$	-931	-153	31	220	133	124	
	$1e-5$	-4,732	-117	125	112	116	121	
	$1e-6$	-47,222	-3,005	137	113	39	12	
	$1e-7$	-774	-22	111	113	2	0	



(a) (b)

Figure 8: In (a) the WIS estimates of $E_{q_i(x)}[X^2]$ diverges from the true values as $q_i(x)$ diverges from $p(x)$. However (b) shows that the relative ordering based on the WIS estimates is reliable.

Table 5: Hyperparameters used for TNPG experiments.

Total environment steps	5e6
Timesteps per iteration	10,000
Policy	MLP
– Number of fully connected layers	2
– Number of units per layer	64
– Activation	tanh
Baseline TRPO	
– KL constraint	0.01
– Discounting γ	0.99
– GAE-λ	0.98
HOOF specific hyperparameters	
– Search bounds for ϵ	[0.001, 0.1]
– Search bounds for (γ)	[0.85, 1]
– Search bounds for (λ)	[0.85, 1]
– Number of random samples for ϵ	50
– Number of random samples for (γ, λ)	50