Fast Efficient Hyperparameter Tuning for Policy Gradients

Supratik Paul [1] **Vitaly Kurin** [2] **Shimon Whiteson** [1]

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. This makes them unsuitable for real life applications where sample efficiency is paramount. In this paper, we propose Hyperparameter Optimisation on the Fly (HOOF), a gradient-free meta-learning algorithm that requires no more than one training run to automatically learn an optimal schedule for hyperparameters 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 & Hinton, 2012) and ADAM (Kingma & Ba, 2014) have narrow ranges for good learning rates (Henderson et al., 2018b). Even for second order methods like Natural Policy Gradients (NPG) (Kakade, 2001) or Trust Region Policy Optimisation (TRPO) (Schulman et al., 2015), which tend to be more robust to the KL divergence constraint (which can be interpreted as a learning rate), significant performance gains can often be obtained by tuning these parameters (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., 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) 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

[1]Department of Computer Science, University of Oxford [2]Department of Engineering Science, University of Oxford. Correspondence to: Supratik Paul <supratik.paul@cs.ox.ac.uk>.

Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36^{th} International Conference on Machine Learning, Long Beach, California, USA, 2019.

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.

Furthermore, when reward is a sum of multiple separately observed reward streams (van Seijen et al., 2017), HOOF can learn different hyperparameter schedules for each reward stream, leading to even faster and better learning.

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. 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. Finally, we consider tasks with multiple reward streams and show that HOOF enables faster learning in such settings.

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. We abuse notation to use π to denote both the policy as well as the parameters. These methods perform an update of the form

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

Here $f(\psi, \pi)$ represents a step along the gradient direction for some objective function estimated from a batch of trajectories $\{\tau_1^\pi, \tau_2^\pi, \ldots, \tau_K^\pi\}$ which have been sampled using policy π, and ψ is the set of hyperparameters. Since we are interested in optimising ψ, for ease of notation we will drop π from $f(\psi, \pi)$ and write it as $f(\psi)$ for the rest of the paper.

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

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

where α is the learning rate, $A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + ...)$ 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:

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

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 learn a schedule for the hyperparameters 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 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 $p(\tau|\pi) = p(s_0) \prod_{i=0}^T \pi(a_i|s_i) p(s_{i+1}|s_i, a_i)$, 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 second order 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. 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 $KL(\pi_{n+1}||\pi_n)$:

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

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 to (4) is found.

Algorithm 1 HOOF

input Initial policy π_0, number of policy iterations N, search space for ψ, KL constraint ϵ if using first order policy gradient method.

for n = 0, 1, 2, 3, . . . , N **do**
  Sample trajectories $\tau_{1:K}$ using π_n.
  for z = 1, 2, . . . Z **do**
    Generate candidate hyperparameter $\{\psi_z\}$ from the search space.
    Compute candidate policy π^z using ψ_z in (1)
    Estimate $J(\pi^z)$ using WIS (5)
    Compute $KL(\pi^z||\pi_n)$ if using first order policy gradient method,
  end for
  Select ψ_n, and hence π_{n+1}, according to (7) for first order methods, or (4) for second order methods
end for

3.2. HOOF with Multiple Reward Streams

In some environments, the reward function is a sum of multiple reward streams, i.e., $r_t = r_t^1 + r_t^2 + ... + r_t^L$ (van Seijen et al., 2017). For example, if we are trying to learn a locomotion behaviour for a robot there could be a reward stream for forward motion, another that penalises joint movement, and another for reaching the goal. If each of these reward streams is observable, we can use HOOF to learn hyperparameters specific to each reward stream. In this setting, the GAE in (2) is simply a linear combination of the advantages for each reward stream, each with its own set of (γ, λ) parameters:

$$A_t^{GAE(\gamma,\lambda)} = A_{t,1}^{GAE(\gamma_1,\lambda_1)} + \ldots + A_{t,L}^{GAE(\gamma_L,\lambda_L)}. \quad (8)$$

3.3. Greedy Maximisation

Setting the hyperparameters at each iteration with HOOF requires greedily maximising $J(\pi_{n+1})$ by solving (4). This can be done using random search or BO, depending on the computational expense of generating and evaluating each candidate π_n.

In (2), $g(\gamma, \lambda)$ is independent of α. Thus, if we only want to learn a schedule for α, the gradient $g(\gamma, \lambda)$ needs to be computed once. Subsequently, computing π_{n+1} for different α involves a multiplication and an addition operation, which is far cheaper than computing the gradient. Thus, in this case we can employ random search to solve (4) efficiently.

If we use HOOF to learn (γ, λ) as well, g_n has to be computed for each setting of (γ, λ). With neural net value function approximations, 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.

A computationally expensive step arises when using second order methods combined with deep neural net policies with tens of thousands of parameters. If the policy had few enough parameters that $I(\pi)^{-1}$ can be computed exactly and stored in memory, then an update to π_n can be computed efficiently. To work with large policies, TRPO uses the conjugate gradient method to approximate $I(\pi)^{-1}g$ directly without explicitly computing $I(\pi)^{-1}$. When used with NPG, the resulting algorithm is referred to as Truncated Natural Policy Gradients (TNPG) (Duan et al., 2016). This implies that each setting of (γ, λ) considered requires a new run of the conjugate gradient algorithm. Thus, BO might be a suitable choice in such situations. However, our experiments suggest that random search with a rather small sample size of 10 performs well even in this case.

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 & 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 & 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, and batch size. 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 KL constraint hyperparameter for first order methods, this do not affect sample efficiency, as shown in Section 5.1.

Other work on non-gradient based methods includes that of Kearns & 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 & 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 & 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 compare HOOF's performance with two baselines. 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.

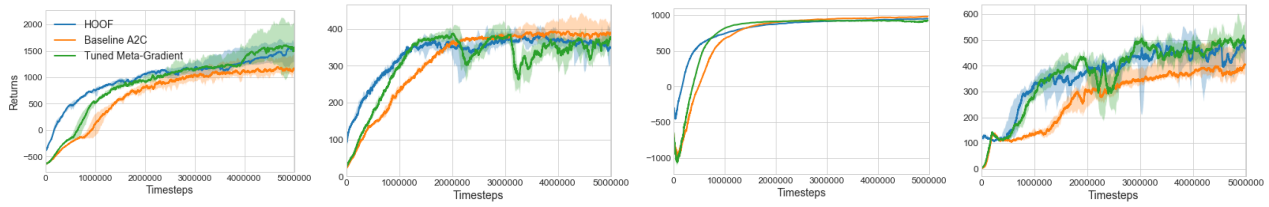
5.1. HOOF with A2C

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

$$f(\alpha) = \alpha\{\nabla_\theta \log \pi_\theta(a|s)(R - V_\theta(s)) + c_1 \nabla_\theta (R - V_\theta(s))^2 + c_2 \nabla_\theta H(\pi_\theta(s))\}, \quad (9)$$

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.



(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

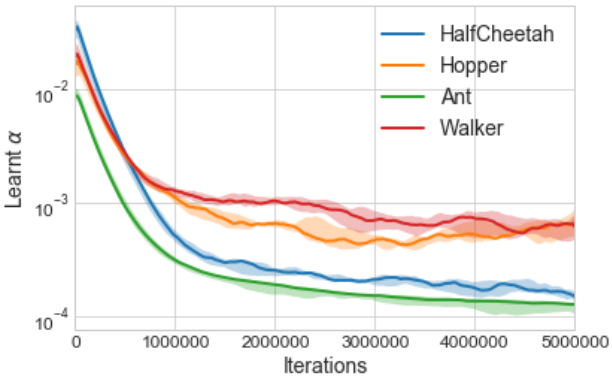


Figure 2. Schedule for the learning rates learnt by HOOF. Refer to Equations (9).

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 learning rates learnt by HOOF are presented in Figure 2.

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.

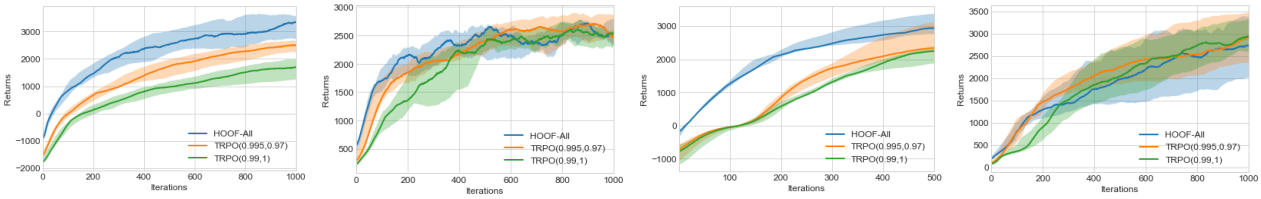
To show that HOOF's performance is robust to ϵ, its own hyperparameter quantifying the KL constraint, we repeated our experiments with different values of ϵ. The results presented in Table 1 show that HOOF's performance is stable across different values of this parameter. This is not surprising – the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

Appendix A.2 contains further experimental details, including results confirming that the KL constraint is crucial to ensuring sound WIS estimates.

In Appendix A.3 we also show that HOOF is 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.

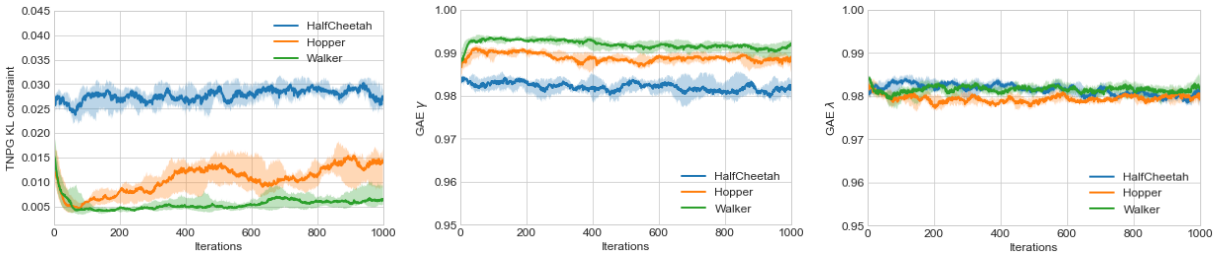
5.2. HOOF with TNPG

A major disadvantage of second order 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 and TRPO address this by



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

Figure 3. Performance of HOOF-All vs TRPO baselines.



(a) HOOF-All: learnt δ (b) HOOF-All: learnt γ (c) HOOF-All: learnt γ

Figure 4. Hyperparameter schedule learnt by HOOF-All for HalfCheetah, Hopper, and Walker. The y-axes shows the bounds of the respective search spaces.

using the conjugate gradient algorithm to efficiently compute $I(\pi)^{-1}g$. 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-All), and compare it to two baseline versions of TRPO: one with $\gamma = 0.99$ and $\lambda = 1$ following Schulman et al. (2015); Duan et al. (2016), and another with $\gamma = 0.995$ and $\lambda = 0.97$ following Henderson et al. (2018a); Rajeswaran et al. (2017). We refer to these as TRPO(0.99,1) and TRPO(0.995,0.97). The KL constraint for both is set to 0.01 following Schulman et al. (2015); Henderson et al. (2018a).

Figure 3 shows the learning curves of HOOF-all and the two TRPO baselines. Across all four environments TRPO(0.995,0.97) outperforms TRPO(0.99,1). HOOF-All learns much faster, and achieves a significantly better return than TRPO(0.995,0.97) in HalfCheetah and Ant. In Hopper and Walker, there is no significant performance difference between HOOF and the TRPO(0.995,0.97) baseline. However, the large variation in the returns for all methods in Walker suggests that the choice of the random seed has a far greater impact on performance than the choice of the hyperparameters.

Figure 4 presents the learnt $(\delta, \gamma, \lambda)$ for HalfCheetah, Hopper, and Walker. The results show that different KL constraints and GAE hyperparameters are needed for different domains. See Appendix C.2 for plots of the learnt hyperparameters for Ant.

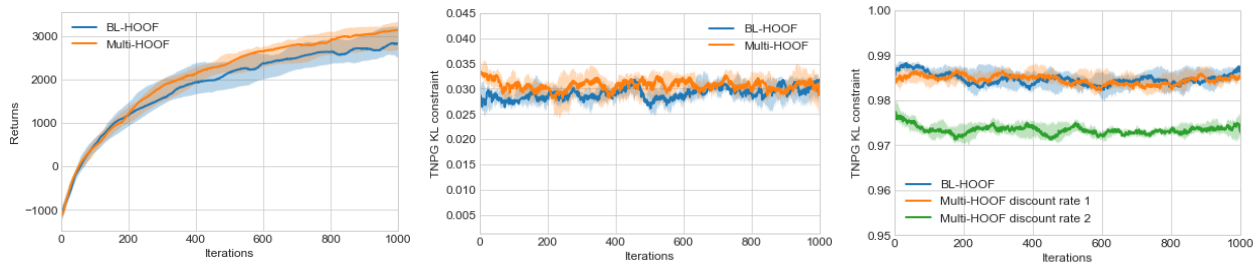
5.3. Tasks with Multiple Reward Streams

The reward function for HalfCheetah is the sum of two components: positive reward for forward movement and penalties for joint movements. Similarly for Ant, the agent gets a fixed reward at each timestep it survives, together with other rewards for forward movement and penalties for joint movement. These additive reward components can be exposed to the learning agent through Gym's API. We use this to test if learning a separate set of hyperparameters for each reward stream with HOOF can improve learning.

Following (8), our method Multi-HOOF, learns a single KL constraint, but a different γ for each reward stream. We compare Multi-HOOF to a second HOOF baseline (BL-HOOF) that learns a single KL constraint and γ for the full reward function. For both λ was fixed to 1.

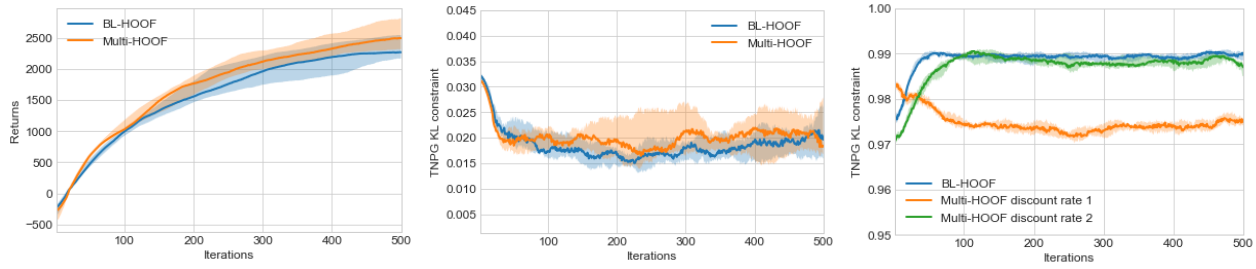
Figure 5 present the results, together with the learnt hyperparameters for HalfCheetah. They show that Multi-HOOF learns a significantly different discount rate for each reward stream and that this helps improve performance.

The results for Ant presented in Figure 6, show that once again learning multiple discount schedules helps improve performance. In Figure 6c, Multi-HOOF's discount rate schedule corresponding to the forward movement reward stream starts lower than that of BL-HOOF's discount rate, but the two soon converge. The discount rate for the survival



(a) Returns (b) Learnt KL constraints (c) Learnt discount rates

Figure 5. Performance and hyperparameters learnt by Multi-HOOF and BL-HOOF for each reward stream in HalfCheetah with multiple reward streams.



(a) Returns (b) Learnt KL constraints (c) Learnt discount rates

Figure 6. Performance and hyperparameters learnt by Multi-HOOF and BL-HOOF for each reward stream in Ant with multiple reward streams.

reward stream on the other hand starts high and then reduces. We believe this is because during early stages of training the key signal for policy improvement comes from the survival reward stream as the agent learns to stay alive. Once it starts getting the survival reward consistently, the forward movement reward stream provides a much better signal for learning a better policy.

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

While this paper focused on learning only a few hyperparameters of policy gradient methods, in principle HOOF could be used to learn other hyperparameters as well, e.g., those of Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016) or Model Agnostic Meta-Learning (MAML) (Finn et al., 2017), which could lead to more stable learning.

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

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

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

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

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

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

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

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

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

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

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

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

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

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

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

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016.

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

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

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

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

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

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

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

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

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

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tieleman, T. and Hinton, G. 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, 2012.

van Seijen, H., Fatemi, M., Laroche, R., Romoff, J., Barnes, T., and Tsang, J. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

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

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

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

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

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 2.

Table 2. 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	
– 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
HOOF specific hyperparameters	
– Initial search bounds for α	[0,1e-2]
– Number of random samples for α	100

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. Performance of HOOF without KL constraint

Figure 7 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.3. Robustness to 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 SGD instead. The learning curves presented in Figure 8 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 (9)). 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 trade-off 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 3. 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.

C.1. Implementation details

Our codebase for the TNPG experiments is based on RLLab (Duan et al., 2016) implementation. The hyperparameters are presented in Table 4.

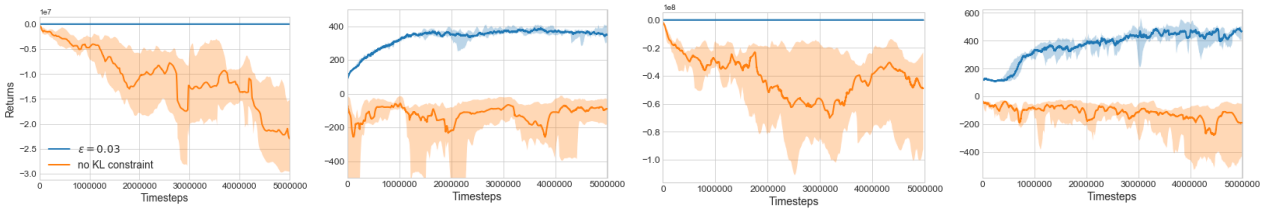
Table 3. Results of grid search over meta-gradients hyperparameters. * denotes algorithm diverged with returns $< -10^5$. Settings with returns greater than HOOF returns are shown in bold.

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	

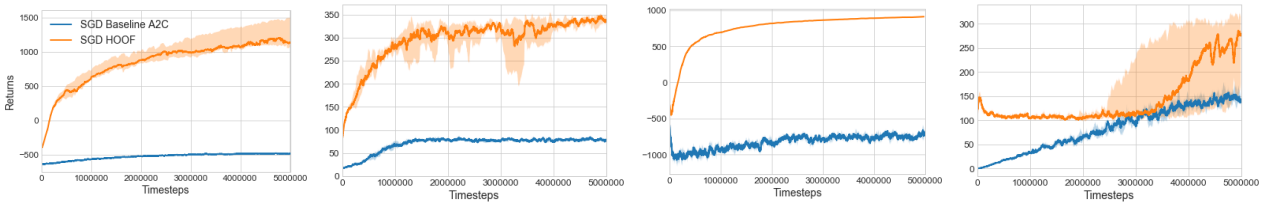
Ant		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	*	393	*	*	*	*	952
	$1e-3$	*	761	752	950	926	884	
	$1e-4$	-47,393	-156	687	672	666	655	
	$1e-5$	*	375	588	-595	-739	-692	
	$1e-6$	*	283	373	-1,081	-1,073	-1,006	
	$1e-7$	-1,257	-514	-361	-1,017	-1,042	-964	

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

Figure 7. Comparison of the performance of HOOF with $\epsilon = 0.03$ and HOOF without any KL constraint.



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

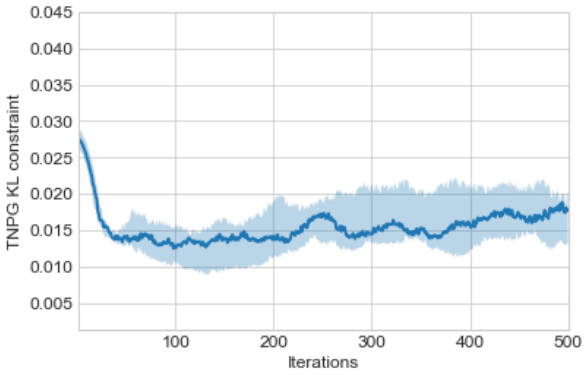
Figure 8. Comparison of the performance of HOOF with $\epsilon = 0.03$ and Baseline A2C where the optimiser is SGD (instead of RMSProp).

Table 4. Hyperparameters used for TNPG experiments.

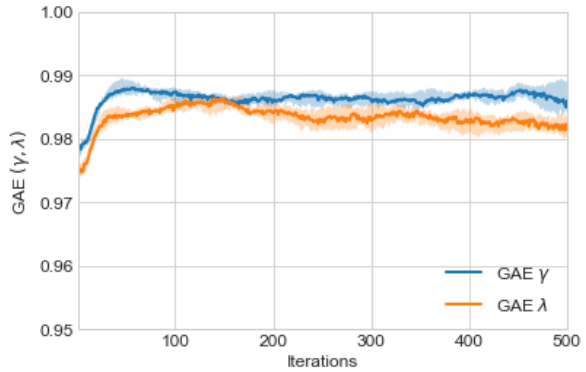
Batch size	
– HalfCheetah-v1	20
– Hopper-v1	20
– Walker2d-v1	40
– Ant-v1	40
Baseline TRPO	
– KL constraint	0.01
– Discounting γ	0.99 or 0.995
– GAE-λ	1.0 or 0.97
Policy	MLP
– num of fully connected layers	2
– num of units per layer	100
– activation	ReLU
HOOF specific hyperparameters	
– search bounds for ϵ	[0.00125, 0.045]
– search bounds for (γ)	[0,95, 1]
– search bounds for (λ)	[0.95, 1]

C.2. Learnt hyperparameter schedules for Ant

The hyperparameter schedule learnt by HOOF-All for Ant are presented in Figure 9.



(a) HOOF-All: learnt δ



(b) HOOF-All: learnt (γ, λ)

Figure 9. Hyperparameter schedule learnt by HOOF-All for Ant.