Discovered Policy Optimisation

Chris Lu[*] University of Oxford
Jakub Grudzien Kuba[*] [†] UC Berkeley
Alistair Letcher aletcher.github.io
Luke Metz Google Brain
Christian Schroeder de Witt University of Oxford
Jakob Foerster University of Oxford

Abstract

The last decade has been revolutionary for *reinforcement learning* (RL) — it can now solve complex decision and control problems. Successful RL methods were handcrafted using mathematical derivations, intuition, and experimentation. This approach has a major shortcoming—*it results in specific solutions to the RL problem, rather than a protocol for discovering efficient and robust methods.* In contrast, the emerging field of *meta-learning* provides a toolkit for automatic *machine learning* method optimisation, potentially addressing this flaw. However, black-box approaches which attempt to discover RL algorithms with minimal prior structure have thus far not been successful. *Mirror Learning*, which includes RL algorithms, such as PPO, offers a potential framework. In this paper we explore the Mirror Learning space by meta-learning a "drift" function. We refer to the result as *Learnt Policy Optimisation* (LPO). By analysing LPO we gain original insights into policy optimisation which we use to formulate a novel, closed-form RL algorithm, *Discovered Policy Optimisation* (DPO). Our experiments in *Brax* environments confirm state-of-the-art performance of LPO and DPO, as well as their transfer to unseen settings.

1 Introduction

Recent advancements in deep learning have allowed reinforcement learning (RL) algorithms to successfully tackle large-scale problems. Early deep RL algorithms, such as A2C [22] and DDPG [19] follow the scheme of *generalized policy iteration* (GPI), where the RL agent alternates between policy evaluation and policy improvement phases [31]. Namely, they train a *critic* neural network that estimates the value function, and an *actor* network which models the agent's policy, and use these to estimate the update direction with policy gradients of the expected return. Another approach is that of *trust-region learning* (TRL) methods, which construct a surrogate objective that they optimise within a small region around the current policy [28], aiming at guaranteeing stability of the policy iterates. Indeed, TRL-inspired algorithms, such as TRPO [28] and PPO [29], have been among the most widely used methods [2] and are known for their performance and stability. Nevertheless, although these research threads have delivered a handful of successful techniques, their design relies on concepts handcrafted by humans, rather than discovered in a *learning process*. As a possible consequence, these methods often suffer from various flaws, such as the brittleness to hyperparameter settings [28, 11], and a lack of optimality guarantees.

Unfortunately, the most promising alternative approach, *algorithm discovery*, thus far has been a tough nut to crack. First of all, techniques that enable optimisation of algorithm components in the *outer loop* process—meta RL [27, 7]—are at an early stage of development, and are vastly more

[*] Equal Contribution
[†] Work done while at University of Oxford

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

compute intensive than the RL problem in the *inner loop* [35]. Second, to discover more complex algorithms, one needs to solve a more sophisticated, and computationally taxing meta-optimisation problem. As a result, most methods discover simple algorithms at best [23].

Recently, *Mirror Learning* [16], a new theoretical framework, introduced an infinite space of provably correct algorithms, all of which share the same template. In a nutshell, a Mirror Learning algorithm is defined by four attributes, but in this work we focus on the *drift function*. A drift function guides the agent's update, usually by penalising large changes. Any mirror learning algorithm provably achieves monotonic improvement of the return, and converges to an optimal policy [16]. Popular RL methods such as TRPO and PPO are instances of Mirror Learning.

In this paper, we use meta-learning to *discover* a new state-of-the-art (SOTA) RL algorithm within the Mirror Learning space. Our algorithm thus inherits theoretical convergence guarantees by construction. Specifically we parameterise a drift function with a neural network, which we then meta-train using *evolution strategies* [26, ES]. The outcome of this meta-training is a specific mirror-learning algorithm which we name *Learnt Policy Optimisation* (LPO). However, we take a step further and learn from LPO about important RL concepts that it has discovered by visualising and analysing its optimisation objective.

While having a neural network representation of a novel, high-performing drift function is a great first step, our next goal is to *understand* the relevant algorithmic features of this drift function.

Building upon these insights we propose a new, closed-form algorithm which we name —*Discovered Policy Optimisation* (DPO). We evaluate LPO and DPO in the *Brax* [8] continuous control environments, where they obtain superior performance compared to PPO. Importantly, both LPO and DPO generalise to environments that were not used for training. To our knowledge, DPO is the first theoretically-sound, deep RL algorithm that was discovered via meta-learning.

2 Related Work

In the last few years, significant effort has been directed at developing effective RL algorithms through both algorithmic and implementational advances. Fujimoto et al. [9] combine DDPG policy training with estimates of pessimistic Bellman targets from a separate critic. Hsu et al. [14] stabilise the, previously unsuccessful [29], KL-penalised version of PPO and improve its robustness through novel policy design choices. Haarnoja et al. [12] introduce a mechanism that automatically adjusts the temperature parameter of the *entropy bonus* in SAC. However, none of these hand-crafted efforts succeeds in fully mitigating common RL pathologies, such as sensitivity to hyperparameter choices and lack of domain generalisation [4]. This motivates radically expanding the RL algorithm search space through automated means.

Indeed, recent work explores applications of meta-RL techniques in guiding RL algorithm discovery and design. RL^2 equips a learning agent with a recurrent neural network conditioning on transitions between tasks, and adapts the agent's behaviour to the current environment [4]. Similarly, a MAML agent learns policy meta-parameters which can adapt to any task with a few steps of gradient descent [7]. Neither RL^2 nor MAML, however, go beyond improving the robustness of ultimately hand-crafted algorithms. To overcome this, Xu et al. [35] introduce an actor-critic method that adjusts its hyperparameters online using *meta-gradients* that are updated with every few inner iterations. Similarly, STAC [37] uses implementation techniques from IMPALA [5] and auxiliary loss-guided meta-parameter tuning to further improve on FRODO.

Such advances have inspired extending meta-gradient RL techniques to more ambitious objectives, including the discovery of algorithms *ab initio*. Notably, Oh et al. [23] succeeded in meta-learning an RL algorithm, LPG, that can solve simple tasks efficiently without explicitly relying on concepts such as value functions and policy gradients. Similarly, Evolved Policy Gradients [13, EPG] meta-trains a policy loss network function with Evolution Strategies [26, ES]. Although EPG surpasses PPO in average performance, it suffers from much larger variance [13]. MetaGenRL [15] instead meta-learns the loss function for deterministic policies which are inherently less affected by estimators' variance [30]. MetaGenRL, however, fails to improve upon DDPG [19] in terms of performance, despite building up on it. Neither EPG nor MetaGenRL have resulted in the discovery of novel analytical RL algorithms, perhaps due to the limited interpretability of the loss functions learnt. Lastly, Co-Reyes et al. [3], Garau et al. [10] and Alet et al. [1] discover and improve standard RL conventions by evolving,

symbolically, algorithms represented as graphs, which leads to improved performance in simple tasks. However, none of those trained-from-scratch methods inherit correctness guarantees, limiting our certainty of the generality of their abilities. In contrast our method, LPO, is meta-developed in a Mirror Learning space [16], where every algorithm is guaranteed convergence to an optimal policy. As a result to this construction, meta-training of LPO is easier than that of methods that learn "from scratch", and achieves great performance across environments. Furthermore, thanks to the clear meta-structure of Mirror Learning, LPO is interpretable, and lets us discover new learning strategies. This lets us introduce DPO—an efficient algorithm with a closed-form formulation that exploits the discovered learning concepts.

3 Background

In this section, we introduce the essential concepts required to comprehend our contribution—the RL and Meta-RL problem formulations, as well as the Mirror Learning and Evolution Strategies frameworks for solving them, respectively.

3.1 Reinforcement Learning

Formulation We formulate the reinforcement learning (RL) problem as a *Markov decision process* (MDP) [31] represented by a tuple $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma, d\rangle$ which defines the experience of a learning agent as follows: at time step $t \in \mathbb{N}$, the agent is at state $\mathrm{s}_t \in \mathcal{S}$ (where $\mathrm{s}_0 \sim d$) and takes an action $\mathrm{a}_t \in \mathcal{A}$ according to its stochastic policy $\pi(\cdot|\mathrm{s}_t)$, which is a member of the policy space Π. The environment then emits the reward $R(\mathrm{s}_t, \mathrm{a}_t)$ and transits to the next state s_{t+1} drawn from the transition function, $\mathrm{s}_{t+1} \sim P(\cdot|\mathrm{s}_t, \mathrm{a}_t)$. The agent aims to maximise the expected value of the total discounted return,

$$\eta(\pi) \triangleq \mathbb{E}[\mathrm{R}^\gamma|\pi] = \mathbb{E}_{\mathrm{s}_0\sim d, \mathrm{a}_{0:\infty}\sim\pi, \mathrm{s}_{1:\infty}\sim P}\Big[\sum_{t=0}^{\infty}\gamma^t R(\mathrm{s}_t, \mathrm{a}_t)\Big]. \tag{1}$$

The agent guides its learning process with value functions that evaluate the expected return conditioned on states or state-action pairs

$$V_\pi(s) \triangleq \mathbb{E}[\mathrm{R}^\gamma|\pi, \mathrm{s}_0 = s] \quad \text{(the state value function)},$$
$$Q_\pi(s, a) \triangleq \mathbb{E}[\mathrm{R}^\gamma|\pi, \mathrm{s}_0 = s, \mathrm{a}_0 = a] \quad \text{(the state-action value function)}.$$

The function that the agent is concerned about most is the *advantage function*, which computes relative values of actions at different states,

$$A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s). \tag{2}$$

Policy Optimisation In fact, by updating its policy simply to maximise the advantage function at every state, the agent is guaranteed to improve its policy, $\eta(\pi_{\text{new}}) \geq \eta(\pi_{\text{old}})$ [31]. This fact, although requiring a maximisation operation that is intractable in large state-space settings tackled by deep RL (where the policy π_θ is parameterised by weights θ of a neural network), has inspired a range of algorithms that perform it approximately. For example, A2C [22] updates the policy by a step of policy gradient (PG) ascent

$$\theta_{k+1} = \theta_k + \frac{\alpha}{B}\sum_{b=1}^{B} A_{\pi_{\theta_k}}(s_b, a_b)\nabla_\theta \log \pi_{\theta_k}(a_b|s_b), \quad \alpha \in (0, 1), \tag{3}$$

estimated from a batch of B transitions. Nevertheless, such simple adoptions of *generalized policy iteration* [31, GPI] suffer from large variance and instability [38, 30, 29]. Hence, methods that constrain (either explicitly or implicitly) the policy update size are preferred [28]. Among the most popular, as well as successful ones, is *Proximal Policy Optimization* [29, PPO], inspired by *trust region learning* [28], which updates its policy by maximising the PPO-clip objective,

$$\pi_{k+1} = \arg\max_{\pi\in\Pi} \mathbb{E}_{\mathrm{s}\sim\rho_{\pi_k}, \mathrm{a}\sim\pi_k}\Big[\min\Big(\frac{\pi(\mathrm{a}|\mathrm{s})}{\pi_k(\mathrm{a}|\mathrm{s})}A_{\pi_k}(\mathrm{s}, \mathrm{a}), \text{clip}\big(\frac{\pi(\mathrm{a}|\mathrm{s})}{\pi_k(\mathrm{a}|\mathrm{s})}, 1 \pm \epsilon\big)A_{\pi_k}(\mathrm{s}, \mathrm{a})\Big)\Big], \tag{4}$$

where the $\text{clip}(\cdot, 1 \pm \epsilon)$ operator clips (if necessary) the input so that it stays within $[1 - \epsilon, 1 + \epsilon]$ interval. In deep RL, the maximisation oracle in Equation (4) is implemented through a few steps of gradient ascent on policy parameters.

Meta-RL The above approaches to policy optimisation rely on human-possessed knowledge, and thus are limited by humans' understanding of the problem. The goal of *meta-RL* is to optimise the algorithm of the learning agent. Formally, suppose that an RL algorithm $\texttt{alg}_\phi$, parameterised by ϕ, trains an agent for K iterations. Meta-RL aims to find the meta-parameter $\phi = \phi^*$ such that the expected return of the output policy, $\mathbb{E}[\eta(\pi_K)|\texttt{alg}_\phi]$, is highest.

3.2 Mirror Learning

A mirror-learning agent [16], in addition to value functions, has access to the following operators: the *drift function* $\mathfrak{D}_{\pi_k}(\pi|s)$ which, intuitively, evaluates the significance of change from policy π_k to π at state s; the *neighbourhood operator* $\mathcal{N}(\pi_k)$ which forms a region around the policy π_k; as well as sampling and drift distributions $\beta_{\pi_k}(\mathrm{s})$ and $\nu^{\pi}_{\pi_k}(\mathrm{s})$ over states. With these defined, a mirror-learning algorithm updates an agent's policy by maximising the mirror objective

$$\pi_{k+1} = \arg\max_{\pi \in \mathcal{N}(\pi_k)} \mathbb{E}_{\mathrm{s}\sim\beta_{\pi_k}}\big[A_{\pi_k}(\mathrm{s},\mathrm{a})\big] - \mathbb{E}_{\mathrm{s}\sim\nu^{\pi}_{\pi_k}}\big[\mathfrak{D}_{\pi_k}(\pi|\mathrm{s})\big]. \tag{5}$$

If, for all policies π and π_k, the drift function satisfies the following conditions:

1. It is non-negative everywhere and zero at identity $\mathfrak{D}_{\pi_k}(\pi|s) \geq \mathfrak{D}_{\pi_k}(\pi_k|s) = 0$,
2. Its gradient with respect to π is zero at $\pi = \pi_k$,

then the Mirror Learning algorithm attains the monotonic improvement property, $\eta(\pi_{k+1}) \geq \eta(\pi_k)$, and converges to the optimal return, $\eta(\pi_k) \to \eta(\pi^*)$, as $k \to \infty$ [16]. A mirror-learning agent can be implemented in practice by specifying functional forms of the drift function and neighbourhood operator, and parameterising the policy of the agent with a neural network, π_θ. As such, the agent approximates the objective in Equation (5) by sample averages, and maximises it with an optimisation method, like gradient ascent. It can be shown that PPO is a valid instance of Mirror Learning, with the drift function:

$$\mathfrak{D}^{\text{PPO}}_{\pi_k}(\pi|s) \triangleq \mathbb{E}_{\mathrm{a}\sim\pi_k}\Big[\mathrm{ReLU}\Big(\Big[\frac{\pi(\mathrm{a}|s)}{\pi_k(\mathrm{a}|s)} - \mathrm{clip}\big(\frac{\pi(\mathrm{a}|s)}{\pi_k(\mathrm{a}|s)}, 1\pm\epsilon\big)\Big]A_{\pi_k}(s,\mathrm{a})\Big)\Big]. \tag{6}$$

Although in its formulation, PPO puts no explicit constraints on its update size [29], as its maximisation oracle (see Equation (5)) is N steps of gradient ascent, with learning rate α and gradient clipping threshold c, it implicitly employs a neighbourhood of an Euclidean ball or radius $N\alpha c$ around θ_k.

Different mirror-learning algorithms can display different properties, such as sample complexity, world-clock time efficiency, or they can aid fulfilling other purposes [16]. That is to say, depending on a preference, a different algorithm is optimal, and PPO is not necessarily the one. In this paper, we specify the optimality criterion to be the return of the Kth iterate, $\eta(\pi_K)$, and discover a mirror-learning algorithm that is superior to PPO with respect to it.

3.3 Evolution Strategies

Evolution Strategies [25, 26, ES] is a backpropagation-free approach to optimisation of stochastic functions. At their core lies the following identity, which holds for any continuously differentiable function F of ϕ, and any positive scalar σ

$$\nabla_\phi \mathbb{E}_{\epsilon\sim N(0,I)}[F(\phi+\sigma\epsilon)] = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim N(0,I)}[F(\phi+\sigma\epsilon)\epsilon], \tag{7}$$

where $N(0,I)$ denotes the standard multivariate normal distribution. By taking the limit $\sigma \to 0$, the gradient on the left-hand side recovers the gradient of $\nabla_\phi F(\phi)$. These facts inspire an approach of optimising F with respect to ϕ without estimating gradients with backpropagation—for a random sample $\epsilon_1, \ldots, \epsilon_n \sim N(0,I)$, the vector $\frac{1}{n\sigma}\sum_{i=1}^{n} F(\phi+\sigma\epsilon_i)\epsilon_i$ is an unbiased gradient estimate. To reduce variance of this estimator, antithetic sampling is commonly used [24]. In the context of Meta-RL, where ϕ is the meta-parameter of an RL algorithm $\texttt{alg}_\phi$, the role of $F(\phi)$ is played by the average return after the training, $F(\phi) = \mathbb{E}[\eta(\pi_K)|\phi]$. As oppose to the meta-gradient approaches described in Section 2, ES does not require backpropagation of the gradient through the whole training episode—a cumbersome procedure which, often approximated by the truncated backpropagation, introduces bias [33, 34, 23, 6, 21].

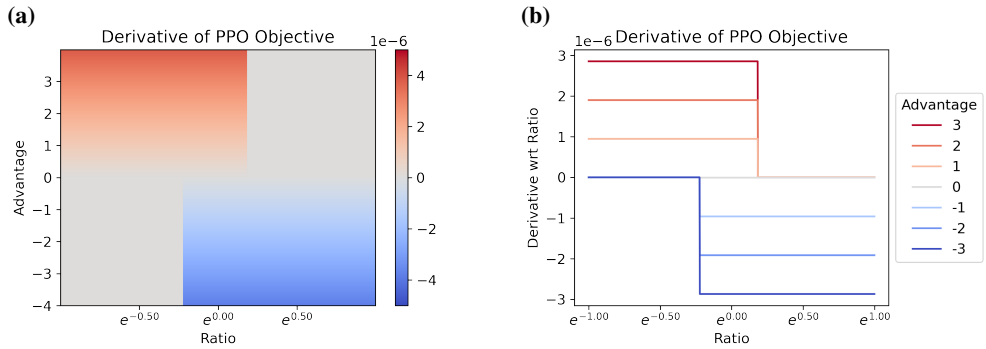


Figure 1: PPO objective visualisation: **(a)** is the heat map of the ratio derivative of the PPO objective, and **(b)** shows its slices for fixed advantage values. The algorithm encourages updates towards actions with positive values of the ratio derivative.

4 Methods

Our overall approach is to *meta-learn* a drift function to perform policy optimisation over a fixed episode length K. Hence, our meta-objective is the expected final return,

$$F(\phi) = \mathbb{E}[\eta(\pi_K)|\phi].$$

In Subsection 4.1 we describe the parameterisation of our learnt drift function, while in Subsection 4.2 we provide a detailed description of our meta-learning approach.

4.1 Drift Function Network

The drift function that we learn takes form $\mathfrak{D}_{\pi_k}(\pi|s) = \mathbb{E}_{\mathrm{a}\sim\pi_k}[f_\phi(\mathbf{x})|s]$, where $f_\phi(\boldsymbol{x})$ as a fully-connected neural network parameterised by ϕ. Our drift network is a function of the probability ratio between a candidate and the old policy, $r = \pi(a|s)/\pi_k(a|s)$, and of the advantage $A = A_{\pi_k}(s, a)$ (which we assume to be normalised across each batch). To ease learning complicated mappings, we include non-linear transformations of these arguments, ultimately forming the following input

$$\boldsymbol{x}_{r,A} = [(1-r),\ (1-r)^2,\ (1-r)A,\ (1-r)^2A,\ \log(r),\ \log(r)^2,\ \log(r)A,\ \log(r)^2A]\,.$$

In order to guarantee that the neural network is a valid drift function, it suffices to impose that $f_\phi(\boldsymbol{x}_{r,A}) = 0$ and $\nabla_r f_\phi(\boldsymbol{x}_{r,A}) = \mathbf{0}$ whenever $r = 1$, and $f_\phi(\boldsymbol{x}_{r,A}) \geq 0$ everywhere. As in our model, $\boldsymbol{x}_{r,A} = 0$ whenever $r = 1$, the former condition is guaranteed by excluding bias terms from the network architecture. To meet the latter two conditions, we apply the ReLU activation at the last layer with a slight shift, $x \mapsto \mathrm{ReLU}(x - \xi)$, where $\xi = 10^{-6}$.

To alleviate the difficulty of the meta-training, the main variant of our implementation initialises the drift function near the PPO one. That is, before we pass the last layer's output to shifted ReLU, we add to it the PPO drift function (more precisely, the input of expectation in Equation 6). We have found that this operation leads to a better performance and generalisation across different environments. Furthermore, such a setting directly tests for the optimality of the drift function of PPO. If PPO was optimal, the network could simply learn the zero mapping.

In principle, the drift function is not restricted to operate with probability ratios and advantages only; it could also accept other arguments, such as algorithm hyperparameters or statistics measuring the progress of training. However, many of these vary greatly, in the dimensionality or scale, across different instances or types of environments. Furthermore, a large number of arguments would impede the analysis of the learnt drift function, and comparison to PPO, which are also goals of our work. Hence, we work with simple and transferable ratios and normalised advantage estimates. Nevertheless, for specific applications, it may be beneficial to consider other, possibly domain-specific, information, and we do encourage it.

4.2 Meta-Training the Drift Function Network

The meta-objective we optimise is the performance of the learner policy at the end of training: $F(\phi) = \mathbb{E}\big[\eta(\pi_{\theta_K})|\mathtt{alg}_\phi\big]$, where θ_K is the Kth (last) iterate of the RL training under the mirror-

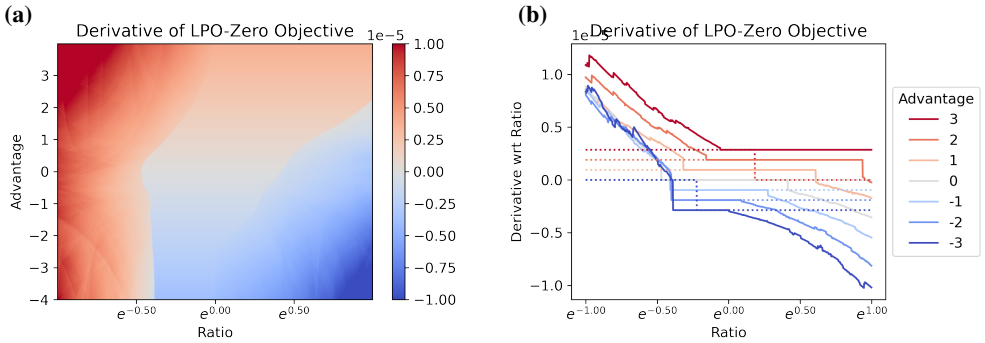


Figure 2: Visualisation of the LPO-Zero (learning from scratch) objective: **(a)** is the heat map of the ratio derivative of the LPO-Zero objective, and **(b)** shows its slices for fixed advantage values. The algorithm encourages updates towards actions with positive values of the ratio derivative.

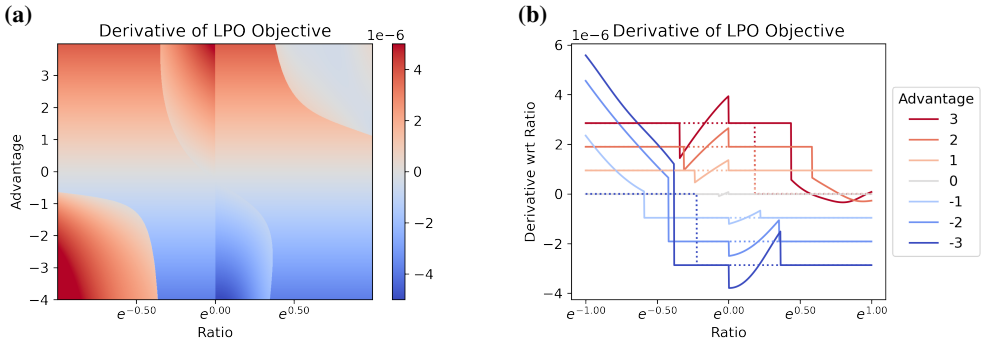


Figure 3: Visualisation of the LPO objective: **(a)** is the heat is the heat map of the ratio derivative of the LPO objective, and **(b)** shows its slices for fixed advantage values. The algorithm encourages updates towards actions with positive values of the ratio derivative.

learning algorithm $\texttt{alg}_\phi$. The expectation is taken over the randomness of the initial parameter θ_0 and stochasticity of the environment. We solve this problem using ES with antithetic sampling. At each generation (outer loop iteration), we sample a batch of perturbations of ϕ, initialise the policy parameters θ_0, and then train the policy under $\texttt{alg}_\phi$, using the drift function's parameter ϕ, for K iterations. At the end of the inner-loop training, we estimate the return of the final policy ϕ_{θ_K}, and use it to estimate the gradient of $\nabla_\phi F(\phi)$ as in Equation (7).

We meta-learn the drift function and evaluate policies trained by it in the *Brax* [8] physics simulator environments. Brax is designed to take advantage of parallel computation on accelerators, allowing us to roll out thousands of episodes in parallel, and train entire RL agents within minutes. Thus, it is well suited for this type of optimisation problem. Furthermore, we increase its efficiency by vectorising the *policy optimisation algorithm itself*, which lets us train hundreds of RL agents per minute using accelerators. We implement our method on top of the Brax version of PPO, which provides a Mirror Learning-friendly code template, keeping the policy architecture and training hyperparameters unchanged. For meta-training we use both *evosax* [17] and the *Learned_optimization* [20] libraries. For full details of meta-training see Appendix A.

5 Empirical Studies

We consider two different meta-training setups. First, as described in Subsection 5.1 we attempt to learn a drift function completely from scratch to investigate how similar it is to existing algorithms like PPO. Second, in Subsection 5.2 we ask whether we can learn a drift function that successfully generalises to multiple environments, if it is initialised near PPO.

5.1 Learning drift functions from scratch

In this setting, f_ϕ is a neural network with two hidden layers of size 256 and a ReLU activation function. We meta-train it across 5 Brax environments. We name the resulting algorithm LPO-Zero, and visualise it in Figure 2.

Interestingly, LPO-Zero appears to have learnt a few PPO-like features, as can be observed on Figure 2. For example, it appears to have learnt to clip the update incentive at a specific ratio threshold, much like PPO; however, it only does so for negative advantages. Nevertheless, LPO-Zero largely underperforms with respect to LPO, and possibly requires much more training to catch up. We present figures with performance evaluation of LPO-Zero in Appendix B.

5.2 Learning with the PPO Initialisation

In this setting, f_ϕ is a small neural network, with a single hidden layer with 128 neurons, with bias terms removed, and $\tanh$ activation function. Here, we add PPO to the output of the last hidden layer, before passing it to the shifted ReLU,

$$f_\phi(\boldsymbol{x}_{r,A}) = \text{ReLU}\Big(\tilde{f}_\phi(\boldsymbol{x}_{r,A}) - \xi + \text{ReLU}\left((r - \text{clip}(r, 1 \pm \epsilon)) \cdot A\right)\Big), \tag{8}$$

where $\tilde{f}_\phi(\boldsymbol{x}_{r,A})$ is the output of the last hidden layer of the drift network. As such, the resulting drift function similar to that of PPO at initialisation.

Surprisingly, we have found that meta-training in a **single** environment is sufficient to generate drift functions whose abilities transfer to unseen tasks. Moreover, we found that the learnt drifts generally display similar characteristics. For readability, we chose the drift function that was trained on Ant, whose induced algorithm we refer to as *Learnt Policy Optimisation* (LPO). Visualisations and results for drift functions trained on other environments can be found in the Appendix C.

The results on Figure 6 show that LPO, trained only on Ant, outperforms PPO in unseen environments. Furthermore, the Brax PPO implementation uses different hyperparameters, such as the number of update epochs and the total number of timesteps, for each of the tasks. This means that LPO, which was trained on *Ant* with hyperparameters associated to it, is robust not only against new environments, but also against new hyperparameters. We visualise the derivative of the LPO loss in Figure 3, which enables us to derive an analytical version of it in Section 6.

6 Analysis of LPO

In this section, we analyse the two key features that are consistently learnt and contribute most to LPO's performance. We then interpret their effect on *policy entropy*, and the *update asymmetry* discovered by LPO, through which it differs largely from PPO (recall Figure 3 for visualisation).

Rollback for negative advantage. In the bottom-left quadrant of the heat map, which corresponds to $A < 0$ and $r < 1$, we observe that the ratio derivative of the LPO objective is positive in a large region, roughly corresponding to $r < 1 - \epsilon$. This implies that actions that fall into this quadrant, although seemingly not appealing, are encouraged to be taken by the agent, which can be interpreted as a form of *rollback* [32]. Hence, LPO learns to decrease r down to $1 - \epsilon$, but unlike PPO, encourages r to stay precisely around that value. By doing so, LPO prevents the agent from giving up on actions that appear poor at the moment, and encourages it to keep exploring them at a moderate frequency.

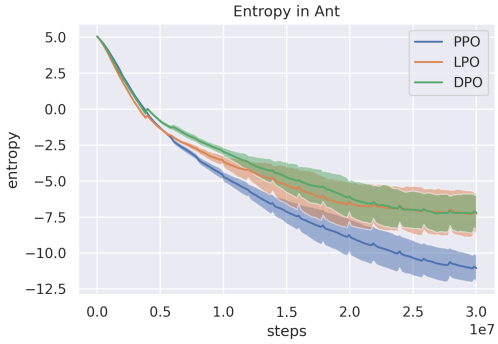


Figure 4: Entropy comparison, throughout training on Ant, between PPO (blue), LPO (orange), and DPO (green, see Section 7) across 10 seeds. Error bars denote standard error. While entropy of all methods decrease throughout training, the entropy of policy learned by both LPO and DPO remain significantly higher than that of PPO.

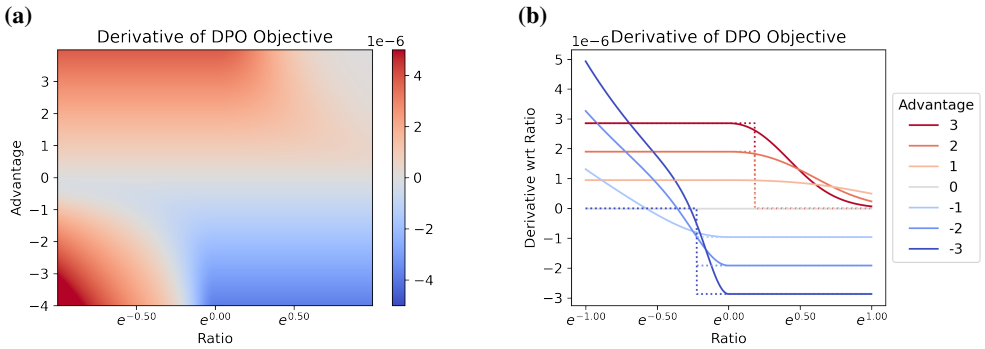


Figure 5: Visualisation of the DPO objective: **(a)** is the heat is the heat map of the ratio derivative of the DPO objective, and **(b)** shows its slices for fixed advantage values. Positive values of the derivative encourage updates towards the action.

Cautious optimism for positive advantage. The upper-right quadrant corresponds to $A > 0$ and $r > 1$, which is induced by actions that seem the most appealing to update to. Nevertheless, LPO is cautious in doing so, gradually decreasing the pace of its update towards them, and eventually abstaining from chasing the most extreme advantage values—these may come from critic errors. We want to highlight that this view of LPO on positive advantages in much more sophisticated than that of PPO, which simply removes any incentive from updating towards actions with $r > 1 + \epsilon$, and thus can be viewed as *optimistic* relative to PPO.

Implicit entropy maximisation. Together, these two central features of LPO encourage the agent to spread its policy probability mass moderately over all actions, thus leading to larger entropy and allowing for richer exploration. Thus, LPO ***has implicitly discovered entropy maximisation***, which we demonstrate in Figure 4.

Update asymmetry. LPO learns *asymmetric* features that respect a natural asymmetry of behaviour change in RL: increasing r for positive advantage A may encourage exploration of a newly-found action *or* strengthen a dominant action, whereas decreasing r for negative A will always discourage exploration of that action *and* strengthen a dominant action. In this context, the two discussed features of LPO make it completely unlike PPO, which clips the update incentives symmetrically around the origin.

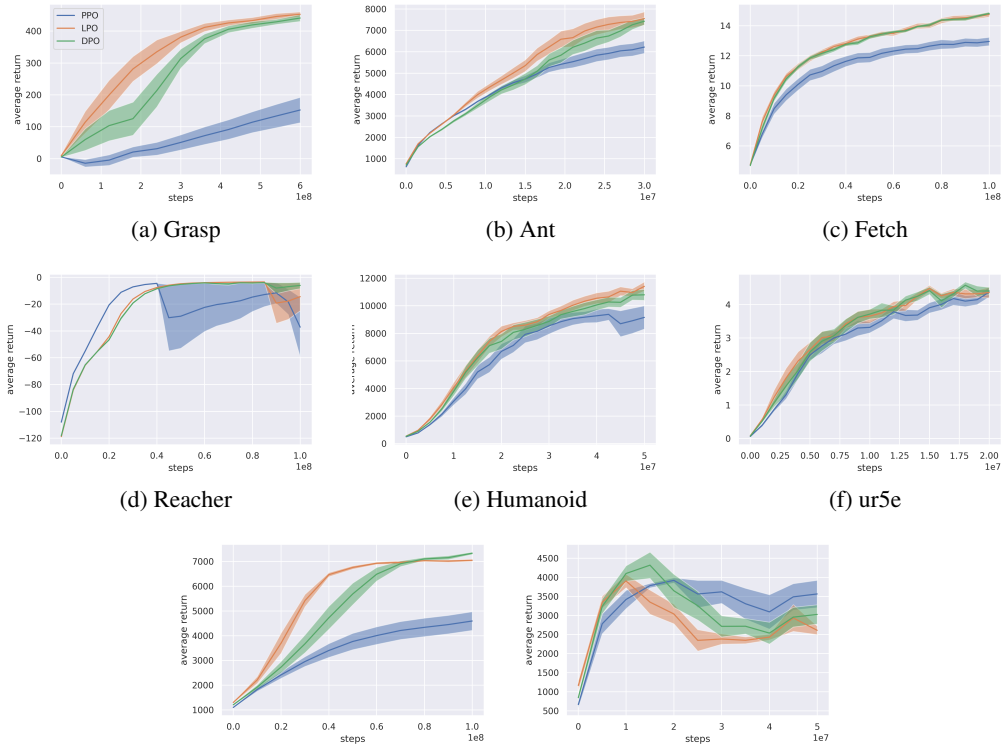
Secondary Features. LPO, but not LPO-Zero, appears to consistently learn objectives with gradient spikes around $r = 1$ in the upper left and lower right quadrants. Nevertheless, adding them to our analytic model of LPO did not improve performance. We speculate, therefore, that these spikes are mostly artifacts of the network parameterisation.

7 Discovered Policy Optimisation: A New RL Algorithm Inspired By LPO

In this section, advancing concepts that LPO has discovered, we introduce a novel algorithm—Discovered Policy Optimization (DPO).

7.1 The Discovered Drift Function Model

Combining the key features identified in Section 6, we construct a closed-form model of LPO that can easily be implemented with just a few lines of code on top of an existing PPO implementation. We name the new algorithm *Discovered Policy Optimisation* (DPO) because we have not derived it—it was instead discovered in the meta-learning process. DPO is a mirror-learning algorithm, with a drift function that takes different functional forms, depending on the sign of advantage A, as dictated by the update asymmetry principle from the previous section. Specifically, we have found that the



(a) Grasp (b) Ant (c) Fetch

(d) Reacher (e) Humanoid (f) ur5e

(g) HalfCheetah (h) Walker2d

Figure 6: Performance comparison between PPO (blue), LPO (orange), and DPO (green) in Brax environments. The curves represent mean evaluation return across 10 random seeds, with error bars showing standard error. Both LPO and DPO beat PPO across most environments even though they were only meta-trained on Ant, and make use of no hyper parameter tuning. Furthermore, both LPO and DPO are more consistent across runs.

(parameter-free) drift function

$$f(r, A) = \begin{cases} \text{ReLU}\big((r-1)A - \alpha \tanh((r-1)A/\alpha)\big) & A \geq 0 \\ \text{ReLU}\big(\log(r)A - \beta \tanh(\log(r)A/\beta)\big) & A < 0 \end{cases} \tag{9}$$

faithfully reproduces the key features of LPO (cautious optimism and rollback) for appropriate constants $\alpha = 2, \beta = 0.6$ (see Appendix D for verification of the drift conditions). We visualise DPO in Figure 5 and note that even the "crossing-over" of gradient slices of LPO on Figure 3 is faithfully reproduced.

7.2 Results

We compare DPO to PPO and LPO on a variety of Brax environments. We use the PPO implementation provided by Brax, which we enhanced with advantage normalisation as we observed it to improve performance across the majority of the environments. Our methods also use this implementation technique. While evaluating DPO, similarly to LPO, we do not re-tune *any* hyperparameters that were originally selected for PPO in Brax. The results on Figure 6 reveal that DPO matches the performance of LPO and outperforms PPO on the evaluated environments, *despite being a two-line analytic model of LPO based on two key features*. This enables RL practitioners to implement DPO as easily as PPO with a performance on par with our best learnt drift function.

8 Conclusion

In this paper, we approached the problem of *algorithm discovery* by restricting our meta-learning to the space of valid *Mirror Learning* algorithms. Specifically, we optimised a drift function parameterised by a neural network, which we trained with *Evolution Strategies*. We consider this work to be an example of the new, promising paradigm of RL algorithm discovery. Namely, our strategy was to develop a high-performing RL algorithm by combining theoretical insights with large-scale computational techniques. As a result of the training, we obtained a theoretically sound method that we named *Learnt Policy Optimisation* (LPO), which outperforms a state-of-the-art baseline (PPO) in unseen environments, and with unseen hyperparameter settings. After analysing the learned features discovered by LPO, we introduced *Discovered Policy Optimisation* (DPO)—a closed-form approximation to LPO. Our experimental results show that DPO matches LPO in performance and robustness to hyperparameter settings. In the future, we plan to expand the variety of inputs to the learnt drift function, as well as to parameterise and meta-learn other attributes of mirror learning. We expect these advancements to provide more insights into policy optimisation, ultimately resulting in more robust and better performing RL algorithms.

References

[1] Ferran Alet, Martin F. Schneider, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Meta-learning curiosity algorithms. *CoRR*, abs/2003.05325, 2020.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[3] John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V Le, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. *arXiv preprint arXiv:2101.03958*, 2021.

[4] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl$ˆ2$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[5] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[6] Xidong Feng, Oliver Slumbers, Ziyu Wan, Bo Liu, Stephen McAleer, Ying Wen, Jun Wang, and Yaodong Yang. Neural auto-curricula in two-player zero-sum games. *Advances in Neural Information Processing Systems*, 34, 2021.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[8] C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[9] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[10] Juan Jose Garau-Luis, Yingjie Miao, John D Co-Reyes, Aaron Parisi, Jie Tan, Esteban Real, and Aleksandra Faust. Multi-objective evolution for generalizable policy gradient algorithms. *arXiv preprint arXiv:2204.04292*, 2022.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[12] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[13] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.

[14] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner, and Moritz Hardt. Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*, 2020.

[15] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[16] Jakub Grudzien Kuba, Christian Schroeder de Witt, and Jakob Foerster. Mirror learning: A unifying framework of policy optimisation. *arXiv preprint arXiv:2201.02373*, 2022.

[17] Robert Tjarko Lange. evosax: Jax-based evolution strategies, 2022.

[18] Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022.

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

[20] Luke Metz, C Daniel Freeman, James Harrison, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Practical tradeoffs between memory, compute, and performance in learned optimizers. *arXiv preprint arXiv:2203.11860*, 2022.

[21] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[22] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[23] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[24] Art B Owen. Monte carlo theory, methods and examples (book draft), 2014.

[25] Ingo Rechenberg. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

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

[27] Jürgen Schmidhuber. On learning how to learn learning strategies. 1995.

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[29] John Schulman, F. Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[30] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[32] Yuhui Wang, Hao He, and Xiaoyang Tan. Truly proximal policy optimization. In *Uncertainty in Artificial Intelligence*, pages 113–122. PMLR, 2020.

[33] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[34] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[35] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[36] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[37] Tom Zahavy, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. A self-tuning actor-critic algorithm. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[38] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *NIPS*, pages 262–270. Citeseer, 2011.

A Meta-Training Details

A.1 LPO-Zero

Our LPO-zero implementation was implemented on top of the learned_optimization library [20]. The drift function is parameterised by a one layer fully connected network with 1 hidden layer and 256 hidden units. Meta-training is done in a distributed fashion using batched, async meta-updates across 350 workers each of which with one TPUv4i accelerator. On a centralized learner process we accumulate gradients from these workers until 350 gradients are computed (using a single perturbation from an antithetic ES based gradient estimator). Once this number is reached, we perform one outer-iteration with Adam using a learning rate of 0.006.

In this experiment, we meta-train over a uniform mixture of the ant, walker2d, halfcheetah and fetch environments. We take the default hparams for PPO from Brax for each implementation except for the number of epochs trained which we set to 183 to match what was done with the ant environment. In each worker, for a particular environment, we perform a full PPO training for both a positive, and negative perturbation of the underlying meta-parameters. At the end of each training, we evaluate 10240 rollouts on the environment with the resulting policy and use these as our fitness function.

Meta-training was done over the course of 2 days and performed approximately 400 outer-updates. We find though performance still increases with increased meta-training time.

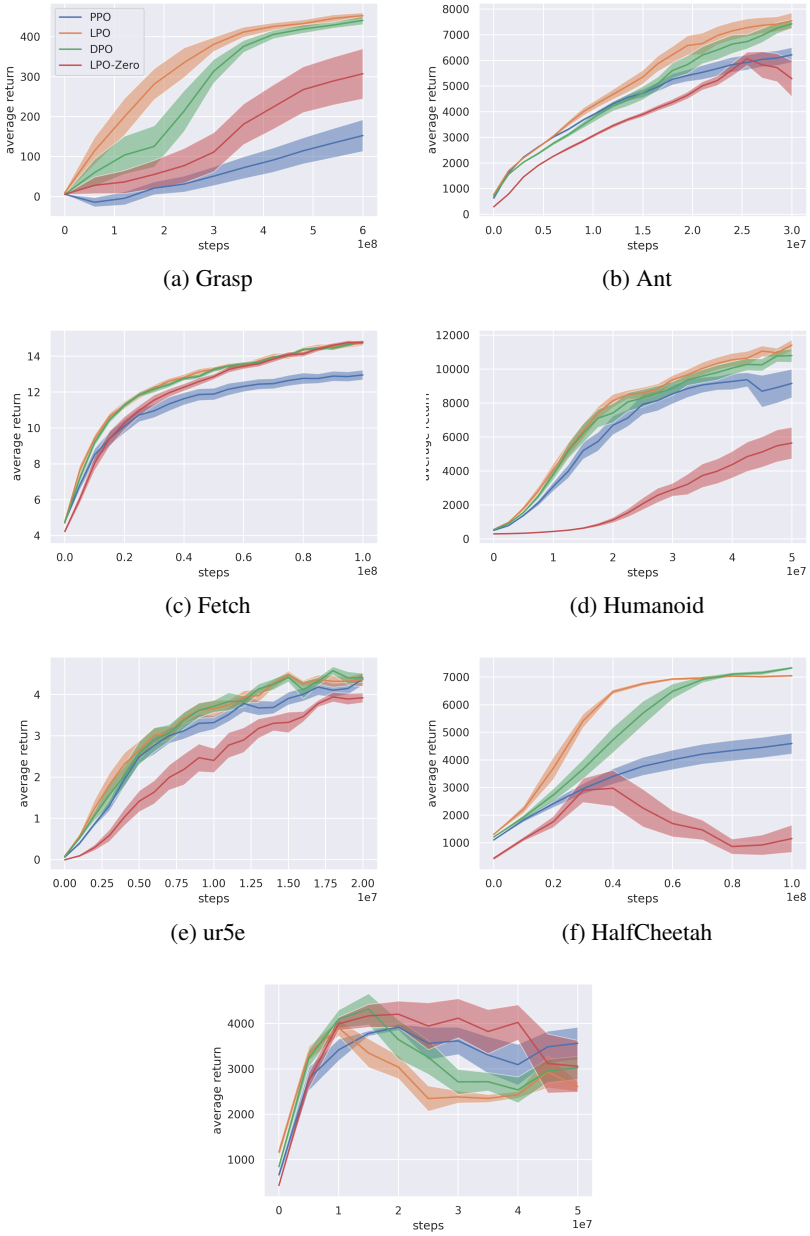
A.2 LPO

Our LPO implementation was implemented on top of the evosax library [17]. The drift function is parameterised by a one layer fully-connected network with 1 hidden layer and 128 hidden units. Meta-training was only done on a single machine with 4 V100 GPU's with synchronous updates. Meta-training was done over the course of 2 days and performed approximately 700 outer-updates. We find though performance still increases with increased meta-training time.

Table 1: Important parameters for Training LPO

Parameter	Value
Population Size	32
Number of Hidden Layers	1
Size of Hidden Layer	128
Number of Generations	672
ES Sigma Init	0.04
ES Sigma Decay	0.999
ES Sigma Limit	0.01
Number of Timesteps	30000000
Unroll Length	5
Number of Minibatches	32
Number of Update Epochs	4
Learning Rate	0.0003
Number of Environments	2048
Batch Size	1024

B LPO-Zero Results



(a) Grasp

(b) Ant

(c) Fetch

(d) Humanoid

(e) ur5e

(f) HalfCheetah

(g) Walker2d

Figure 7: Performance comparison between PPO (blue), LPO (orange), DPO (green), and LPO-Zero (red) in Brax environments. The curves represent mean evaluation return across 10 random seeds, with error bars showing standard error.

C Visualisations of LPO

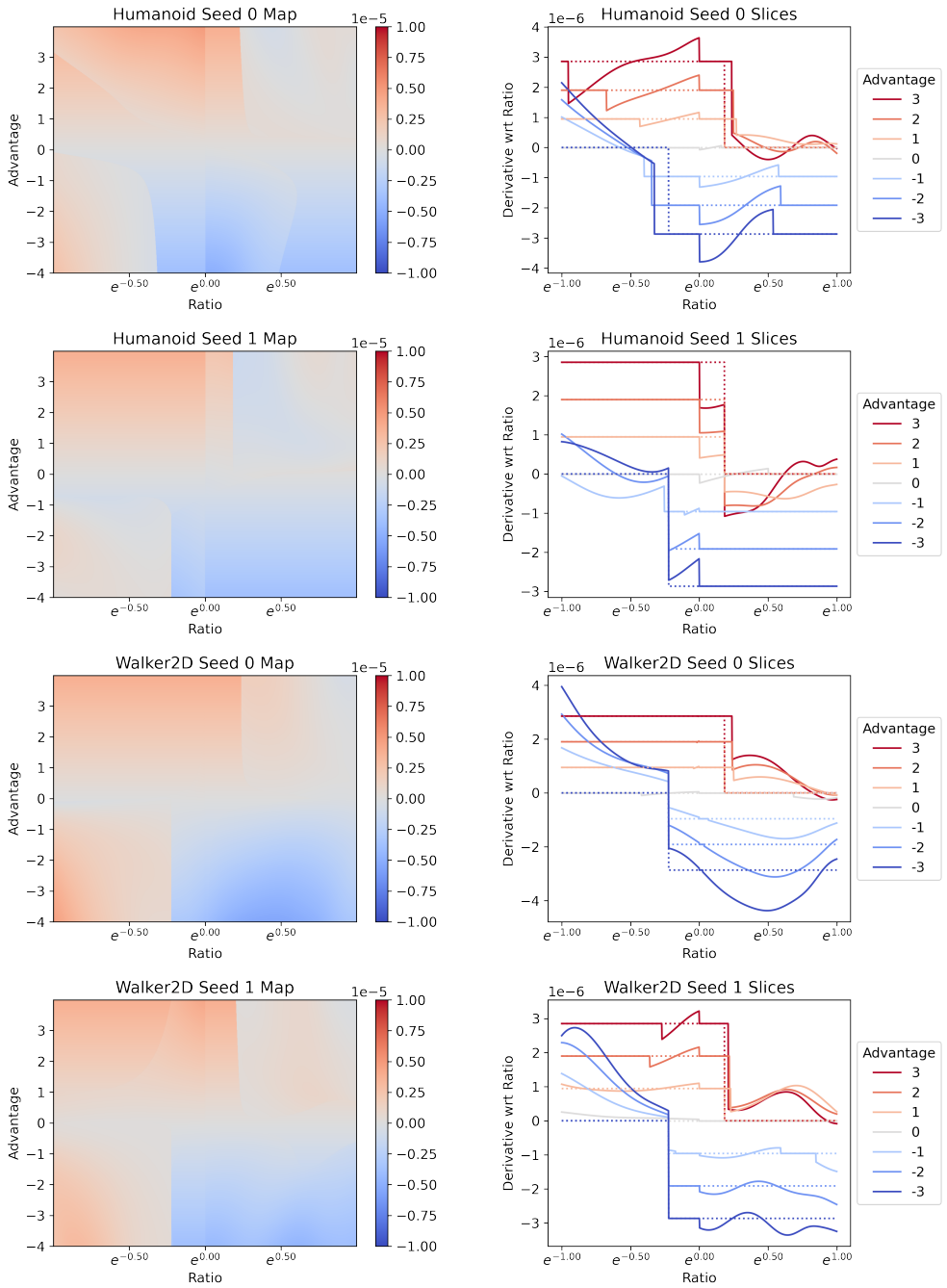


Figure 8: Visualisation of the learned objectives trained on different environments: **(a)** is the heat is the heat map of the ratio derivative of the LPO objective, and **(b)** shows its slices for fixed advantage values. Positive values of the derivative encourage updates towards the action.

D DPO Drift Verification

The DPO drift function is given by

$$f(r, A) = \begin{cases} \text{ReLU}\big((r-1)A - \alpha \tanh((r-1)A/\alpha)\big) & A \geq 0 \\ \text{ReLU}\big(\log(r)A - \beta \tanh(\log(r)A/\beta)\big) & A < 0\,. \end{cases}$$

The first condition for a valid drift is that f be non-negative everywhere, which trivially holds since $\text{ReLU}(x) \geq 0$ for all $x \in \mathbb{R}$.

The second condition is that f be zero at $\pi = \pi_{\text{old}}$. Now $r = \pi/\pi_{\text{old}} = 1$ implies $r - 1 = 0$ and $\log r = 0$, which combined with $\tanh(0) = 0$ imply that $f = 0$ as required.

The final condition is that the gradient of f with respect to π be zero at $\pi = \pi_{\text{old}}$. This is equivalent to having zero gradient with respect to $r = \pi/\pi_{\text{old}}$ at $r = 1$ since the gradients are equal up to a constant. Now writing

$$f^+ = (r-1)A - \alpha \tanh((r-1)A/\alpha) \qquad \text{and} \qquad f^- = \log(r)A - \beta \tanh(\log(r)A/\beta)$$

for $A \geq 0$ and $A < 0$ respectively, we have

$$\frac{\partial f^+}{\partial r} = A - A\cosh^{-2}((r-1)A/\alpha) \qquad \text{and} \qquad \frac{\partial f^-}{\partial r} = \frac{A}{r} - \frac{A}{r}\cosh^{-2}(\log(r)A/\beta)$$

which both evaluate to 0 at $r = 1$, since $\cosh(0) = 1$. This implies for $A \geq 0$ that

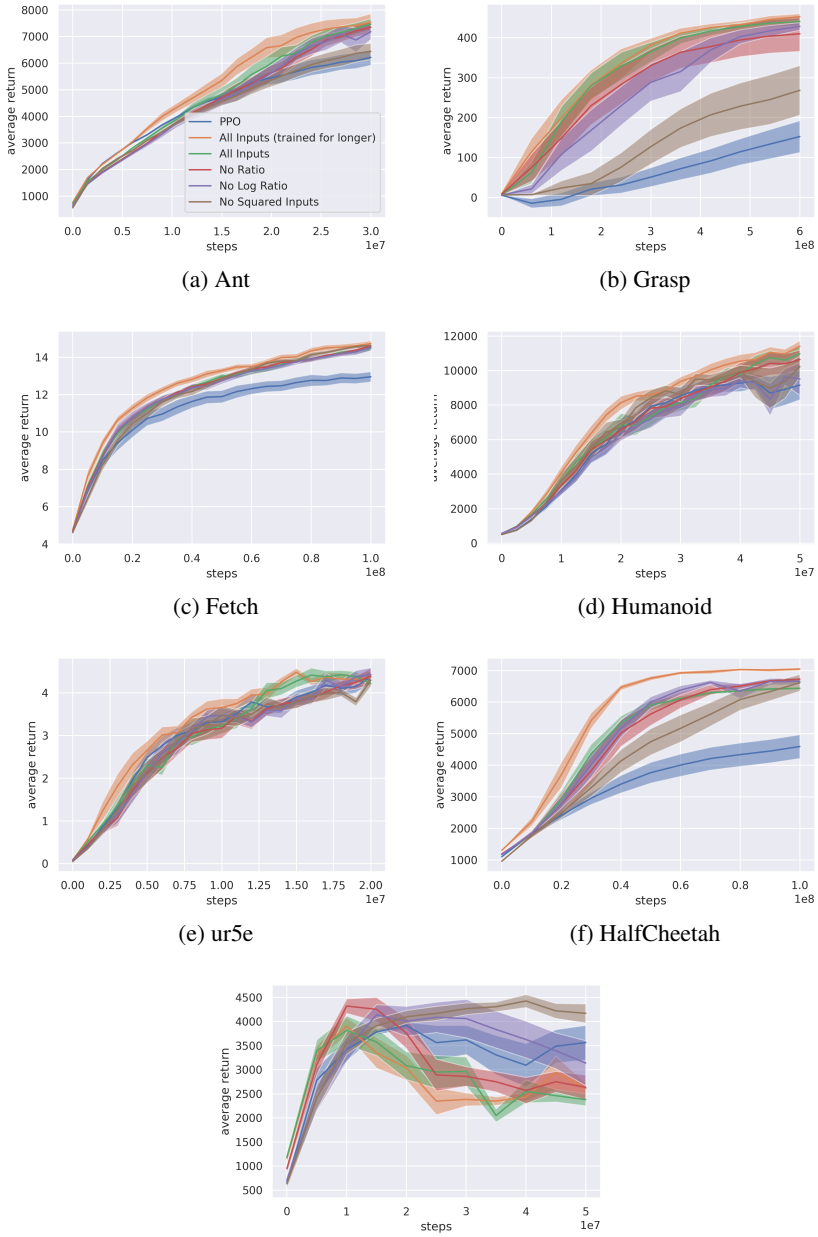
$$\frac{\partial f}{\partial r} = \frac{\partial \text{ReLU}(f^+)}{\partial r} = \begin{cases} \frac{\partial f^+}{\partial r} & \text{if } f^+ \geq 0 \\ 0 & \text{if } f^+ < 0 \end{cases} \quad = \quad 0$$

at $r = 1$ and for $A < 0$ that

$$\frac{\partial f}{\partial r} = \frac{\partial \text{ReLU}(f^-)}{\partial r} = \begin{cases} \frac{\partial f^-}{\partial r} & \text{if } f^- \geq 0 \\ 0 & \text{if } f^- < 0 \end{cases} \quad = \quad 0$$

at $r = 1$. Taken together we conclude, for all A, that f has zero gradient at $r = 1$.

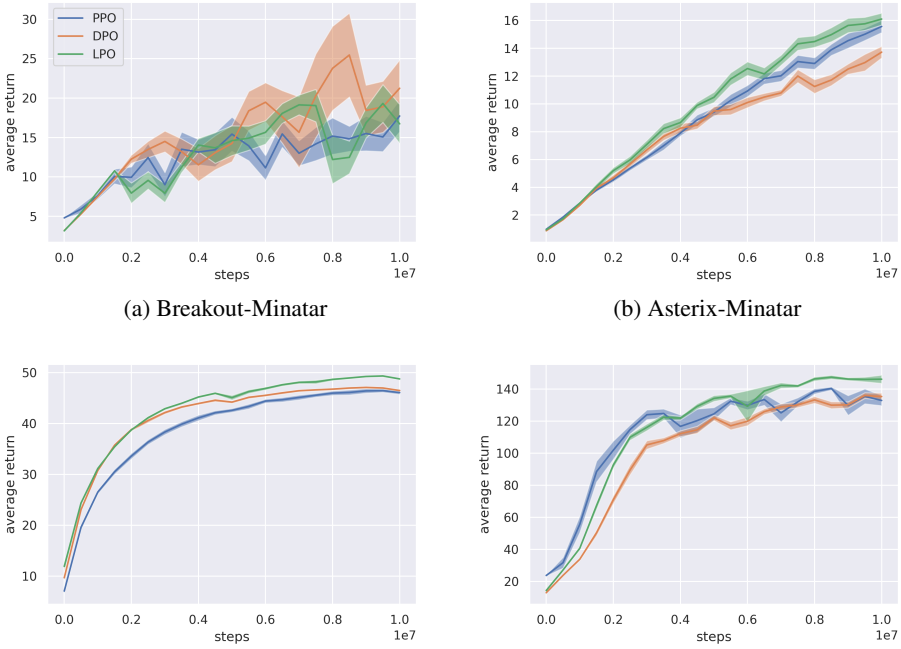
E Ablations on Drift Inputs



(a) Ant (b) Grasp (c) Fetch (d) Humanoid (e) ur5e (f) HalfCheetah (g) Walker2d

Figure 9: Performance comparison between different inputs to the meta-training of the drift function. Note that due to computational constraints, the meta-training was only trained for 208 generations instead of 672. The results show that the meta-trained drift function performs well with respect to multiple possible input parameterisations.

F Minatar Results



(a) Breakout-Minatar

(b) Asterix-Minatar

(c) Freeway-Minatar

(d) SpaceInvaders-Minatar

Figure 10: Performance of PPO, DPO, and LPO on the Minatar environments [18, 36].