Evolved Policy Gradients

Rein Houthooft [1] **Richard Y. Chen** [1] **Phillip Isola** [1] **Bradly C. Stadie** [2] **Filip Wolski** [1]
Jonathan Ho [1] **Pieter Abbeel** [2]

Abstract

We propose a meta-learning approach for learning gradient-based reinforcement learning (RL) algorithms. The idea is to evolve a differentiable loss function, such that an agent, which optimizes its policy to minimize this loss, will achieve high rewards. The loss is parametrized via temporal convolutions over the agent's experience. Because this loss is highly flexible in its ability to take into account the agent's history, it enables fast task learning and eliminates the need for reward shaping at test time. Empirical results show that our evolved policy gradient algorithm achieves faster learning on several randomized environments compared to an off-the-shelf policy gradient method. Moreover, at test time, our learner optimizes only its learned loss function, and requires no explicit reward signal. In effect, the agent internalizes the reward structure, suggesting a direction toward agents that learn to solve new tasks simply from intrinsic motivation.

1. Introduction

Human behavior is motivated by a complex interplay of internal rewards – hunger and thirst, curiosity and amusement (24; 34). This is in stark contrast with how current reinforcement learning (RL) agents are trained. In general, RL agents have no explicit internal rewards. Instead, they are simply given positive and negative feedback from an external teacher. But in many real world scenarios, no such teacher exists. For example, when a human learns to solve a new control task, such as playing the violin, they don't require external rewards to start learning. They immediately have a feel for what to try, and for whether or not they are making progress towards the goal. Effectively, humans have access to very well shaped internal reward functions, derived from prior experience on other motor tasks, or perhaps from listening to and playing other musical instruments. Our aim in this paper is to devise agents that similarly have a prior notion of what constitutes making progress on a novel task. Rather than encoding this knowledge explicitly through memorized behaviors, we encode it implicitly through a learned loss function.

[1]OpenAI [2]UC Berkeley. Correspondence to: Rein Houthooft <rein.houthooft@openai.com>.

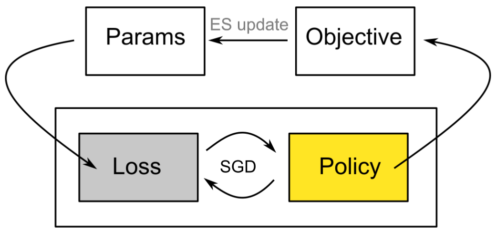


Figure 1: High-level overview of our approach. The method consists of an inner and outer optimization loop. The inner loop (boxed) optimizes the agent's policy against a loss provided by the outer loop, using gradient descent. The outer loop optimizes the parameters of the loss function, such that the learned policy from the inner loop achieves high performance on an arbitrary outer objective, such as solving a control task of interest. The evolved loss L can be understood as a parametrization of policy gradients $\frac{\partial L}{\partial \pi}$, lending the name "evolved policy gradients".

This approach can be seen as a form of meta-learning, in which we learn a learning algorithm. Rather than mining rules that generalize across data points, as in traditional machine learning, meta-learning concerns itself with devising algorithms that generalize across tasks, by infusing prior knowledge of the task distribution. The final result is a method that, via the learned loss, enables quick learning on new tasks drawn from a particular distribution.

Our method consists of two optimization loops. In the inner loop, an agent learns to solve a task, sampled from the task distribution, by minimizing a loss function provided by the outer loop. In the outer loop, the parameters of the loss function are adjusted so as to maximize the final returns achieved after inner loop learning. Figure 1 provides a high-level overview of this approach.

Although the inner loop can be optimized with stochastic gradient descent (SGD), optimizing the outer loop presents substantial difficulty. Each evaluation of the outer objective requires training a complete inner loop agent, and this objective cannot be written as an explicit function of the loss parameters we are optimizing over. Due to the lack of easily exploitable structure in this optimization problem, we turn to evolution strategies (ES) (23; 31; 10; 25) as a blackbox optimizer. The evolved loss L can be viewed as a parametrization of policy gradients $\frac{\partial L}{\partial \pi}$, lending the name "evolved policy gradients".

In addition to encoding prior knowledge, the learned loss offers several advantages compared to current RL methods. Since RL methods optimize for short-term returns instead of accounting for the complete learning process, they may get stuck in local minima and fail to explore the full search space. Prior works add auxiliary reward terms that emphasize exploration (4; 12; 21; 40; 2; 22) and entropy loss terms (20; 29; 9; 15). These terms are often traded off using a separate hyperparameter that is not only task-dependent, but also dependent on which part of the state space the agent is visiting. As such, it is unclear how to include these terms into the RL algorithm in a principled way. Using ES to evolve the loss function allows us to optimize the true objective rather than short-term returns, namely the final trained policy performance. Our learned loss function improves on standard RL algorithms by allowing the loss function to be adaptive to the environment and agent history, leading to faster learning and learning without reward shaping.

Our contributions include the following:

- Formulating a meta-learning approach that learns a differentiable loss function for RL agents.
- Optimizing the parameters of this loss function via ES, overcoming the challenge that final returns are not explicit functions of the loss parameters.
- Designing a loss architecture that takes into account agent history via temporal convolutions.
- Demonstrating that the learned loss function can train agents that achieve higher returns than agents trained via an off-the-shelf policy gradient method.

We set forth the notation in Section 2. Section 3 explains the main algorithm and Section 4 shows its results on several randomized continuous control environments. In Section 5, we compare our methods with the most related ideas in literature. We conclude this paper with a discussion in Section 6.

2. Notation and Background

We model reinforcement learning (38) as a Markov decision process (MDP), defined as the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, p_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action space. The transition dynamic $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}_+$ determines the distribution of the next state s_{t+1} given the current state s_t and the action a_t. $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function and $\gamma \in (0, 1]$ is a discount factor. p_0 is the distribution of the initial state s_0. An agent's policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ generates an action after observing a state.

An episode $\tau \sim \mathcal{M}$ with horizon H is a sequence $(s_0, a_0, r_0, \ldots, s_H, a_H, r_H)$ of state, action, and reward at each timestep t. The discounted episodic return of τ is defined as $R_\tau = \sum_{t=0}^{H} \gamma^t r_t$, which depends on the initial state distribution p_0, the agent's policy π, and the transition distribution T. The expected episodic return given agent's policy π is $\mathbb{E}_\pi[R_\tau]$. The optimal policy π^* maximizes the expected episodic return

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\mathcal{M},\pi}[R_\tau].$$

In high-dimensional reinforcement learning settings, the policy π is often parametrized using a deep neural network $\pi_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. The goal is to solve for $\boldsymbol{\theta}^*$ that attains the highest expected episodic return

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\tau\sim\mathcal{M},\pi_{\boldsymbol{\theta}}}[R_\tau].$$

This objective can be optimized via policy gradient methods (44; 39) by stepping in the direction of $\mathbb{E}[R_\tau \nabla \log \pi(\tau)]$. This gradient can be transformed into a surrogate loss function

$$L_{\text{pg}} = \mathbb{E}[R_\tau \log \pi(\tau)] = \mathbb{E}\left[R_\tau \sum_{t=0}^{H} \log \pi(a_t|s_t)\right], \quad (1)$$

such that the gradient of L_{pg} equals the policy gradient. Through variance-reduction techniques including actor-critic algorithms (13), the loss function L_{pg} is often changed into

$$L_{\text{ac}} = \mathbb{E}\left[\sum_{t=0}^{H} A(s_t) \log \pi(a_t|s_t)\right],$$

that is, the log-probability of taking action a_t at state s_t is multiplied by an advantage function $A(s_t)$.

However, this procedure remains limited since it relies on a particular form of discounting the returns, and taking a fixed gradient step with respect to the policy. Our approach learns a loss rather than using a hand-defined function such as L_{ac}. Thus, it may be able to discover more effective surrogates for making fast progress toward the ultimate objective of maximizing final returns.

3. Methodology

Our meta-learning approach aims to learn a loss function L_ϕ that outperforms the usual policy gradient loss by taking into account the agent's history. This loss function consists of temporal convolutions over the agent's recent history. In addition to internalizing environment rewards, this loss could, in principle, have several other positive effects. For example, by examining the agent's history, the loss could incentivize desirable extended behaviors, such as exploration. Further, the loss could perform a form of system identification, inferring environment parameters and adapting how it guides the agent as a function of these parameters (e.g., by adjusting the effective learning rate of the agent).

The loss function parameters ϕ are evolved through ES and trains agent's policy $\pi(\theta)$ in an on-policy fashion via gradient descent.

3.1. Meta-learning Objective

In our meta-learning setup, we assume access to a distribution $p(\mathcal{M})$ over MDPs. Given a sampled MDP $\mathcal{M}$, the inner loop optimization problem is to minimize the loss L_ϕ with respect to the agent's policy π_θ:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\tau\sim\mathcal{M},\pi_\theta}[L_\phi(\pi_\theta, \tau)]. \tag{2}$$

Note that this is the same as the usual RL objective (Equation (2)), except that we are optimizing a learned loss L_ϕ rather than directly optimizing R. The outer loop objective is to learn L_ϕ such that an agent policy π_{θ^*} trained with the loss function actually does achieve high expected returns, R, in the MDP distribution:

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}\mathbb{E}_{\tau\sim\mathcal{M},\pi_{\theta^*}}[R_\tau]. \tag{3}$$

3.2. Algorithm

The final episodic return R_τ at evaluation cannot be represented as an explicit function of the loss function L_ϕ, and thus we cannot use gradient-based methods to directly solve Equation (3). Our approach, summarized in Algorithm 1, relies on evolutionary strategies to optimize the loss function in the outer loop.

As described by Salimans et al. (25), ES computes the gradient of a function $F(\phi)$ according to

$$\nabla_\phi \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} F(\phi+\sigma\epsilon) = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)} F(\phi+\sigma\epsilon)\epsilon.$$

Similar formulations also appear in prior works including (36; 32; 16). In our case, $F(\phi) = \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}\mathbb{E}_{\tau\sim\mathcal{M},\pi_{\theta^*}}[R_\tau]$ (Equation (3)),. Note that the dependence on ϕ comes through θ^* (Equation (2)).

Step by step, the algorithm works as follows. At the start of each iteration in the outer loop, we generate a standard multivariate normal vector $\epsilon_i \in \mathcal{N}(0, \mathrm{I})$ with the same dimension as the loss function parameter ϕ for each worker $i \in \{1, \dots, n\}$. The outer loop gives each inner-loop worker a perturbed loss function

$$L_i = L_{\phi+\sigma*\epsilon_i}.$$

Algorithm 1: Evolved Policy Gradients (EPG)

Input: Learning rates δ and α, noise standard deviation σ, random environment distribution $p(\mathcal{M})$
[Outer Loop] for epoch $e = 1, \dots, E$ **do**
  [Inner Loop] for each worker $i = 1, \dots, n$ **do**
    Sample random vector $\epsilon_i \sim \mathcal{N}(\mathbf{0}, \mathrm{I})$ and calculate the loss function parameter $\phi + \sigma\epsilon_i$
    Generate a random environment $\mathcal{M}_i$ according to $p(\mathcal{M})$
    for trajectory $j = 1, \dots, K$ **do**
      Sample initial state s_0
      for timestep $t = 0, \dots, M$ **do**
        Sample action a_t from $\pi_\theta(a_t|s_t)$
        Take action a_t, observe reward r_t and next state s_{t+1} from $\mathcal{M}_i$
        If termination signal is reached, reset environment, resampling initial state s_0
      Update policy parameter θ based on the loss function $L_{\phi+\sigma\epsilon_i}$ according to Eq. (4)
    Compute the final return R_i
  Update the parameter ϕ for the loss function L_ϕ according to Eq. (5)

with perturbed parameters $\phi+\sigma*\epsilon_i$ where σ is the standard deviation.

Given a loss functions L_i, $i \in \{1, \dots, n\}$, from the outer loop, each inner loop worker i samples a random MDP from the task distribution, $\mathcal{M}_i \sim p(\mathcal{M})$. The worker then trains a policy π_θ in $\mathcal{M}_i$ over K trajectories $\{\tau_j\}_{j=1}^K$ of M timesteps of experience. Each trajectory may consist of multiple episodes: whenever a termination signal is reached, the environment resets with state s_0 sampled from the initial state distribution $p_0(\mathcal{M}_i)$. After each trajectory, the policy takes a gradient step with respect to minimizing the loss function L_i:

$$\theta \leftarrow \theta - \delta \cdot \nabla_\theta L_i\big(\pi_\theta, \tau_j\big). \tag{4}$$

At the end of the inner-loop training, each worker returns the final return R_i[1] to the outer loop. The outer-loop aggregates the final returns $\{R_i\}_{i=1}^n$ from all workers and updates the loss function parameter ϕ as follows:

$$\phi \leftarrow \phi + \alpha \cdot \frac{1}{n\sigma}\sum\nolimits_{i=1}^{n} R_i\epsilon_i, \tag{5}$$

where α is the outer-loop learning rate.

[1] More specifically, $R_i = R_{i,K} + \frac{1}{K}\sum_{t=0}^{K}\frac{(t+1)}{K}R_{i,t}$ with $R_{i,t}$ the undiscounted return of episode t in worker i.

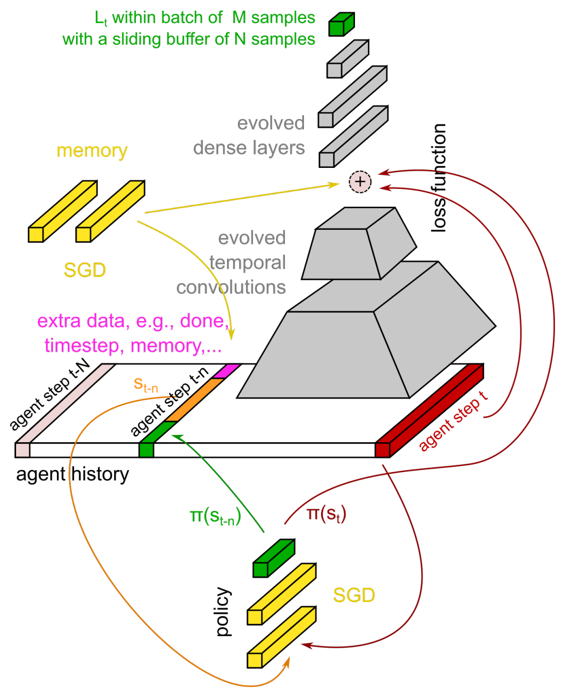


Figure 2: Architecture of a loss computed for timestep t within a batch of M sequential samples (from $t - M$ to t), using temporal convolutions over a buffer of size N (from $t - N$ to t), with $M \leq N$: dense net on the bottom is the policy $\pi(s)$, taking as input the observations (orange), while outputting action probabilities (green). The green block on the top represents the loss output. Gray blocks are evolved, yellow blocks are updated through SGD.

3.3. Architecture

The agent is parametrized using an MLP policy. The agent has a memory unit to assist learning in the inner loop. This memory unit has one layer with constant inputs and the memory parameters are updated during training via SGD in the inner loop but does not directly impact the agent's policy output. Instead, the memory is fed as an input to the loss, which can thereby modulate the learning process. An experience buffer stores the agent's N most recent timesteps of experience, in the form of a tuple (s_t, a_t, r_t, d_t) for each transition.

The loss function $L(\boldsymbol{\phi})$ consists of temporal convolutional layers which generate a context vector f_{context}, and dense layers, which output the loss. The architecture is depicted in Figure 2.

At step t, the dense layers outputs the loss L_t by taking a batch of M sequential samples

$$\{s_i, a_i, d_i, mem, f_{\text{context}}, \pi_{\boldsymbol{\theta}}(\cdot|s_i)\}_{i=t-M}^{t},$$

where $M < N$ and we augment each transition with the memory output mem, a context vector f_{context} generated from the temporal convolutional layers, and the policy distribution $\pi_{\boldsymbol{\theta}}(\cdot|s_i)$. In continuous action space, $\pi_{\boldsymbol{\theta}}$ is a Gaussian policy, i.e. $\pi_{\boldsymbol{\theta}}(\cdot|s_i) = \mathcal{N}(\mu(s_i; \boldsymbol{\theta}), \Sigma(s_i; \boldsymbol{\theta}))$. In discrete action space, $\pi_{\boldsymbol{\theta}}$ represents a multinomial distribution over the discrete actions.

To generate the context vector, we first augment each transition in the buffer with the output of the memory unit and the policy distribution to obtain $\{s_i, a_i, d_i, mem, \pi_{\boldsymbol{\theta}}(\cdot|s_i)\}_{i=t-N}^{t}$. The temporal convolutional layers take them as input and output the context vector f_{context}.

Note that both the temporal convolution layers and the dense layers do not observe the environment rewards directly. However, in special cases, where the reward cannot be fully inferred from the environment, such as the forward-backward random Hopper environment we will examine in Section 4.5, we augment the inputs to the loss function with reward.

During the inner loop training, the parameters for the memory unit is also updated together with the policy. In practice to bootstrap the learning process, we initially add to $L(\boldsymbol{\phi})$ a guidance policy gradient surrogate loss signal, such as the REINFORCE (44) or PPO (30) surrogate loss function L_{pg}, making the total loss

$$\hat{L}_{\boldsymbol{\phi}} = (1 - \alpha) L_{\boldsymbol{\phi}} + \alpha L_{\text{pg}}, \tag{6}$$

with α annealed from 1 to 0 after a finite number of epochs. As such, learning is first derived mostly from the well-structured L_{pg}, while over time $L_{\boldsymbol{\phi}}$ takes over and drives learning completely after $alpha$ is annealed to 0.

4. Experiments

We apply our method to two randomized continuous control MuJoCo environments, namely Hopper (with randomized gravity, friction, body mass, and link thickness) and Reacher (with randomized link lengths). An example of random Hopper and random Reacher is shown in Figure 3.

4.1. Implemention details

In our experiments, the temporal convolutional layers of the loss function has 3 layers. The first layer has a kernel size of 8, stride of 7, and outputs 8 channels. The second layer has a kernel of 4, stride of 2, and outputs 8 channels. The third layer is fully-connected with 32 output units. Leaky ReLu activation is applied to each convolutional layer. The fully connected component takes the trajectory features from the convolutional component concatenated with state, action, reward, termination signal, and policy output as input. It has 1 hidden layer with 16 hidden units and leaky ReLu

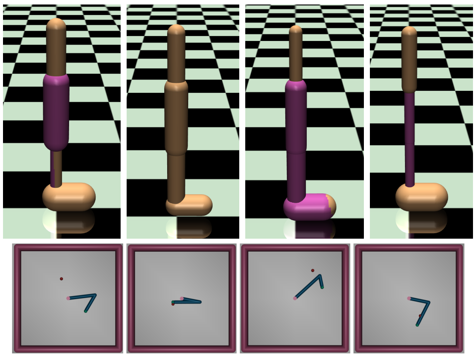


Figure 3: Examples of randomized MuJoCo environments. Top: Hopper with random limb thicknesses. Bottom: Reacher with random limb lengths.

activation, followed by an output layer. The agent's MLP policy has 2 hidden layers of with 64 units with $\tanh$ activation. The memory unit is a 32-unit signal layer with $\tanh$ activation.

For both the PPO as our evolved policy gradient experiments, (inner loop) gradient were performed every 128 sampled time steps, running for 128 updates total for random Hopper and 512 updates for random Reacher. We use $n = 32$ inner-loop workers for both random Hopper and random Reacher in Algorithm 1. PPO was slightly tuned to perform well on these tasks, however, it is important to keep in mind that PPO inherently has trouble optimizing when the number of samples drawn for each batch is low. The buffer size over which the temporal convolutions act in our evolved policy gradient method are the same as the PPO batch size, namely 128. Normalization according to a running mean and standard deviation were applied to all input data (observations, actions, rewards, etc.).

4.2. Is the learned loss effective?

We compare learning performance using learned loss functions against performance using an off-the-shelf policy gradient method, PPO (30). Figure 4 and 5 show learning curves for these two methods on the random Hopper and random Reacher environments respectively.

In both cases, the PPO agent observes reward signals whereas the agent optimizing our learned loss does not observe rewards (note that by test time, α in Eqn. (6) equals 0). Nonetheless, the latter agent learns more quickly and obtains higher returns compared to the PPO agent. This indicates that our method generates an objective that is more effective at training agents, within these task distributions, than off-the-shelf PPO. This is true even though the learned loss does not observe rewards at test time. This demonstrates the potential to use our method when rewards are only available at training time, for example, if a system were trained in simulation but deployed in the real world where reward signals are hard to measure.

4.3. What is the relationship between the learned loss and PPO?

The correlation between the gradients of our learned loss and the PPO objective is around $\rho = 0.5$ (Spearman's rank correlation coefficient) for both Hopper and Reacher. This indicates that the gradients produced by the learned loss differs sufficiently from the PPO objective.

4.4. Does the loss encourage smooth learning?

Figure 6 shows the KL-divergence between policies from one update to the next during the course of training a Hopper, using a randomly initialized loss (left) versus a learned loss (right). After evolution, the policy updates tend to shift the distribution less on each step, but sometimes produce sudden changes, indicated by the spikes.

4.5. Does the loss encourage exploration?

To understand whether the evolved policy gradient algorithm is able to train agents that explore, we test our method and PPO on a specialized forward-backward random Hopper environment: in addition to randomized physics, each sampled Hopper environment either rewards the agent for forward or backward hopping. Note that without observing the reward, the agent cannot infer whether the Hopper environment desires forward or backward hopping. Thus we augment the environment reward to the input batches of the loss function in this setting.

Figure 7 shows learning curves of both PPO agents and agents trained with the learned loss in forward-backward Hopper environment. The learning curves give indication that the learned loss is able to train agents that exhibit exploratory behavior. We see that in most instances, PPO agents stagnate in learning, while agents trained with our learned loss manage to explore both forward and backward hopping and eventually hop in the correct direction. Figure 8 demonstrates the qualitative behavior of our agent during learning.

4.6. How sensitive is the loss to different kinds of inputs?

Our reward-free loss function takes four kinds of inputs: observations, actions, termination signals, and policy outputs, and evaluates entire length N buffers of experience. Which types of input and which time points in the buffer matter the most? In Figure 9, we plot the sensitivity of the learned loss function to each of these kinds of inputs by computing $||\frac{\partial L_{t=25}}{\partial x_t}||_2$ for different kinds of input x_t at different time

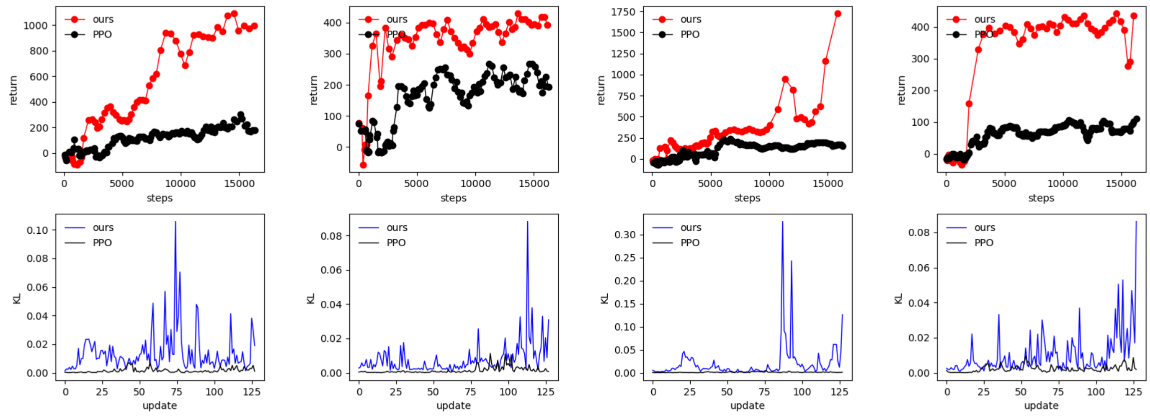


Figure 4: Random Hopper environment learning over 128 (policy updates) $\times 128$ (timesteps per update) $= 16384$ total timesteps: PPO (black) vs no-reward evolved policy gradient algorithm (red). Each subplot column corresponds to a different randomization of the Random Hopper environment. These plots show test time policy learning, after the learned loss was trained for 3000 epochs on Random Hopper, with α in Eq. (6) annealed from 1 to 0 in the first 500 epochs.

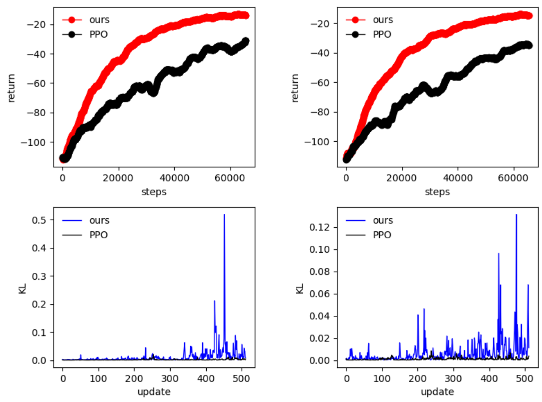


Figure 5: Random Reacher environment learning over 512 (policy updates) $\times 128$ (timesteps per update) $= 65536$ total timesteps: PPO (black) vs no reward evolved policy gradient algorithm (red). The two subplots correspond to two a different randomizations of the Random Reacher environment. These plots show test time policy learning, after the learned loss was trained for 3000 epochs on Random Reacher, with α in Eq. (6) annealed from 1 to 0 in the first 500 epochs.

points t in the input buffer. This analysis demonstrates that the loss is especially sensitive to experience at the current time step where it is being evaluated, but also depends on the entire temporal context in the input buffer. This suggests that the temporal convolutions are indeed making use of the agent's history (and future experience) to score the behavior.

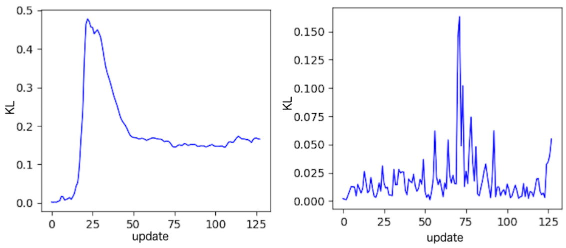
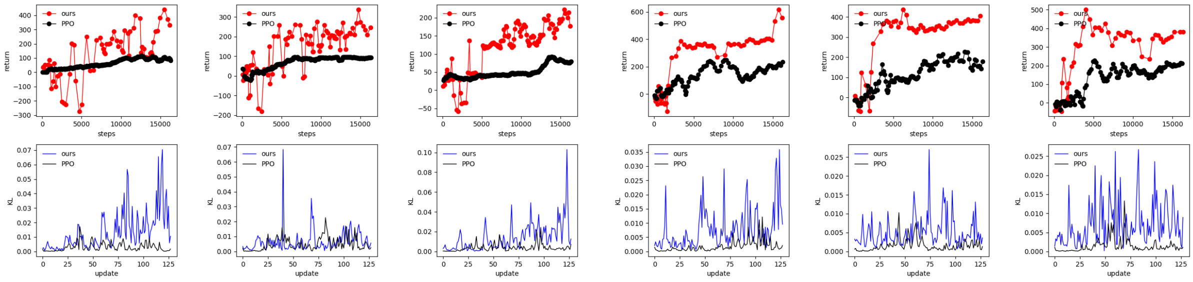


Figure 6: Evolving learning behavior in Random Hopper: the KL-divergence between adjacent policy updates in outer-loop iteration 1 (left) vs the last outer-loop iteration of meta-learning (without PPO guidance signal) (right).

5. Relation to Existing Literature

The concept of learning an algorithm for learning is quite general, and hence there exists a large body of somewhat disconnected literature on the topic.

To begin with, there exist several relevant and recent publications in the meta-learning literature (6; 5; 43). In (6), an algorithm named MAML is introduced. MAML treats the meta-learning problem as in initialization problem. More specifically, MAML attempts to find a policy initialization from which only a minimal number of policy gradient steps are required to solve new tasks. This is accomplished by performing gradient descent on the original policy parameters with respect to the post policy update rewards. While this idea of learning a more universal loss function through meta-learning is similar to ours, their formulation is ultimately much more restrictive and requires differentiability with respect to the objective function.



(a) Random forward Hopper environment.

(b) Random backward Hopper environment.

Figure 7: Forward-backward Hopper environment: besides randomized physics, each Hopper environment randomly decides whether to reward forward or backward hopping. The agent needs to identify whether to jump forward or backwards: PPO (black) vs evolved policy gradient + PPO ($\alpha = 0.5$ in Eq. (6)) (red). Here we can clearly see exploratory behavior, indicated by the negative spikes in the reward curve, the loss forces the policy to try out backwards behavior. Each subplot column corresponds to a different randomization of the environment.

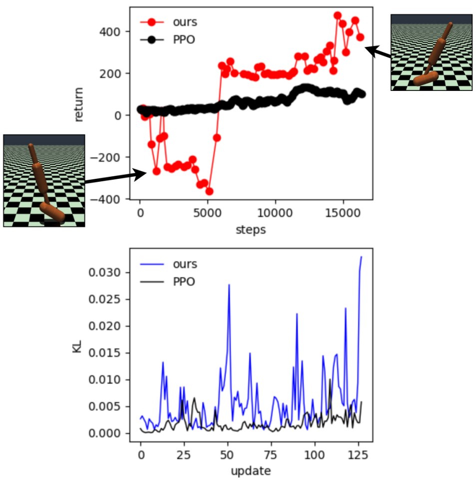


Figure 8: Forward-backward Hopper environment: exploratory behavior of evolved policy gradient algorithm (red) in figuring out the physics settings in comparison to PPO (black). The agent goes backwards for a while, then going forward and exploiting. Inlays indicate qualitative behavior observed in these two phases. We train the evolved policy gradient algorithm for 3000 epochs

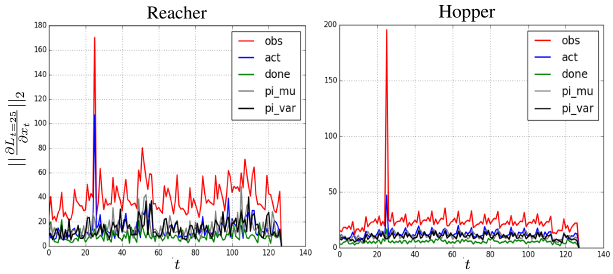


Figure 9: Which inputs is the learned loss function sensitive to? Here we plot the magnitude of the gradient of $L_{t=25}$ w.r.t. different kinds of inputs to the loss function at different time points within the input buffer. Notice that the loss depends highly on the data for the current time point ($t = 25$) but also depends on the length 128 contextual window.

In a work concurrent with ours, Yu et al. (46) extended the model from (7) to incorporate a more elaborate learned loss function. The proposed loss function involves temporal convolutions over trajectories of experience, similar to the method proposed in this work. However, unlike our work, (46) primarily considers the problem of behavioral cloning. Typically, this means their method will require demonstrations, in contrast to our method which does not. Further, their outer objective does not require sequential reasoning and must be differentiable and their inner loop is a single SGD step. We have no such restrictions. Our outer objective is long horizon and non-differentiable and consequently our inner loop can run over tens of thousands of timesteps.

Another recent meta-learning algorithm is RL^2 (5) (which itself is similar to (43)). RL^2 is essentially a GRU policy learning over a task distribution. The GRU receives flags from the environment marking the end of episodes. Using these flags and simultaneously ingesting data for several different tasks, it learns how to compute gradient updates through its internal logic. RL^2 is limited by its decision to couple the policy and learning algorithm (using a GRU for both), whereas we decouple these components. Due to RL^2's policy-gradient-based optimization procedure, we see that it does not directly optimize final policy performance nor exhibit exploration. Hence, extensions have been proposed such as E-RL^2 (37) in which the rewards of episodes sampled early in the learning process are deliberately set to zero to drive exploratory behavior.

Further research on meta reinforcement learning comprises a vast selection. The literature's vastness is further complicated by the fact that the research appears under many different headings. Specifically, there exist relevant literature on: life-long learning, learning to learn, continual learning, and multi-task learning. For example, (27; 26) consider self modifying learning machines (genetic programs). If we consider a genetic program that itself modifies the learned genetic program, we can subsequently derive a meta-GP approach (See (37), for further discussion on how this method relates to the more recent meta learning literature discussed above). The method described above is sufficiently general that it encompass most modern meta-learning approaches. For a further review of other meta learning approaches, see the review articles (33; 41; 42) and citation graph they generate.

Finally, there are several other avenues of related work that tackle slightly different problems. For instance, several methods that attempt to learn a reward function to drive learning. See (3) (which suggests learning from human feedback) and the field of Inverse Reinforcement Learning (17) (which recovers the reward from demonstrations). Both of these fields relate to our ideas on loss function learning. Similarly, (18; 19) apply population-based evolutionary algorithms to reward function learning in gridworld environments. This algorithm is encompass by the algorithms we present in this paper. However, it is typically much easier since learning just the reward function is in many cases a trivial task (e.g., in learning to walk, mapping the observation of distance to a reward function). See also (35) for a different take on evolutionary perspectives of reward learning. Other reward learning methods include the work of Guo et al. (8; 34), which focus on learning reward bonuses. These bonuses are typically designed to augment but not replace the learned reward and have not easily shown to generalize across broad task distributions. Reward bonuses are closely linked to the idea of curiosity, in which an agent attempts to learn an internal reward signal to drive future exploration. Schmidhuber (28) was perhaps the first to examine the problem of intrinsic motivation in a meta-learning context. The proposed algorithms make use of dynamic programming to explicitly partition experience into checkpoints. Further, there is usually little focus on meta-learning the curiosity signal across several different tasks. Finally, the work of (11; 45; 1; 14) studies meta-learning over the optimization process in which meta-learner makes explicit updates to a parametrized model in supervised settings.

6. Discussion

In this paper, we introduced a new meta-learning approach capable of learning a differentiable loss function over thousands of sequential environmental actions. Crucially, this learned loss is both highly adaptive (allowing for quicker learning of new tasks) and highly instructive (sometimes eliminating the need for environmental rewards at test time). In certain cases, the adaptability of our learned loss is appreciated. For example, consider the forward-backward hopper experiments from Section 4.5. Here, the rewards at test time are impossible to infer from observations of the environment alone. Therefore, they cannot be completely internalized. However, if we do get to observe a reward signal on these environments, then we have shown that our algorithm *does* improve learning speed.

Meanwhile, in most other cases, our loss' instructive nature – which allows it to operate at test time without environmental rewards– is interesting and desirable. This instructive nature can be understood as the loss function's internalization of the reward structures it has previously encountered under the training task distribution. We see this internalization as a step toward learning intrinsic motivation. A good intrinsically motivated agent would successfully infer useful actions in new situations by using heuristics it developed over its entire lifetime. This ability is likely required to achieve truly intelligent agents (28).

There exist many interesting directions for future work. Our method demands sequential learning. That is to say, one must first perform outer loop update i before learning about update $i + 1$. This can bottleneck the meta-learning cycle and create large computational demands. Indeed, the number of sequential steps for each inner-loop worker in our algorithm is $E \times K \times M$, using notation from Algorithm 1. In practice, this value may be very high, for example, each inner-loop worker takes approximately 196 million steps to train the loss function used in Figure 5 experiments. Finding ways to parallelize parts of this process, or increase sample efficiency, could greatly improve the practical applicability of our algorithm. Improvements in computational efficiency would also allow the investigation of more challenging tasks. Nevertheless, we feel the success on the environments we tested is non-trivial and provides a proof of concept of our method's power. It is our hope that perhaps one day, reinforcement learning agents will live spectacularly long lives with highly refined internal rewards.

Acknowledgments

We thank Igor Mordatch, Ilya Sutskever, and Karthik Narasimhan for helpful comments and conversations. We thank Maruan Al-Shedivat for assisting with the random MuJoCo environments.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

[2] Richard Y Chen, John Schulman, Pieter Abbeel, and Szymon Sidor. UCB exploration via Q-ensembles. *arXiv preprint arXiv:1706.01502*, 2017.

[3] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4302–4310, 2017.

[4] Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.

[5] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[7] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[8] Xiaoxiao Guo, Satinder Singh, Richard Lewis, and Honglak Lee. Deep learning for reward design to improve monte carlo tree search in atari games. *arXiv preprint arXiv:1604.07095*, 2016.

[9] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

[10] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[11] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001.

[12] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 513–520. ACM, 2009.

[13] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, pages 1008–1014, 2000.

[14] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[15] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2772–2782, 2017.

[16] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[17] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*. Citeseer, 2000.

[18] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

[19] Scott Niekum, Lee Spector, and Andrew Barto. Evolution of reward functions for reinforcement learning. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 177–178. ACM, 2011.

[20] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Pgq: Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

[21] Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

[22] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

[23] I. Rechenberg and M. Eigen. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. 1973.

[24] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

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

[26] Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook. *Diploma thesis, TUM*, 1987.

[27] Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. *Artificial General Intelligence*, 2006.

[28] Juergen Schmidhuber. Exploring the predictable. In *Advances in evolutionary computing*, pages 579–612. Springer, 2003.

[29] John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

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

[31] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*. Birkhäuser, 1977.

[32] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[33] Silver, Yand, and Li. Lifelong machine learning systems: Beyond learning algorithms. *DAAAI Spring Symposium-Technical Report, 2013.*, 2013.

[34] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from.

[35] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

[36] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[37] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, W. Yuhuai, P. Abbeel, and I. Sutskever. Some considerations on learning to explore via meta-reinforcement learning. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2018.

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[39] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

[40] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[41] Taylor and Stone. Transfer learning for reinforcement learning domains: A survey. *DAAAI Spring Symposium-Technical Report, 2013.*, 2009.

[42] Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, 1996.

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

[44] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[45] A Steven Younger, Sepp Hochreiter, and Peter R Conwell. Meta-learning with backpropagation. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3. IEEE, 2001.

[46] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint arXiv:1802.01557*, 2018.