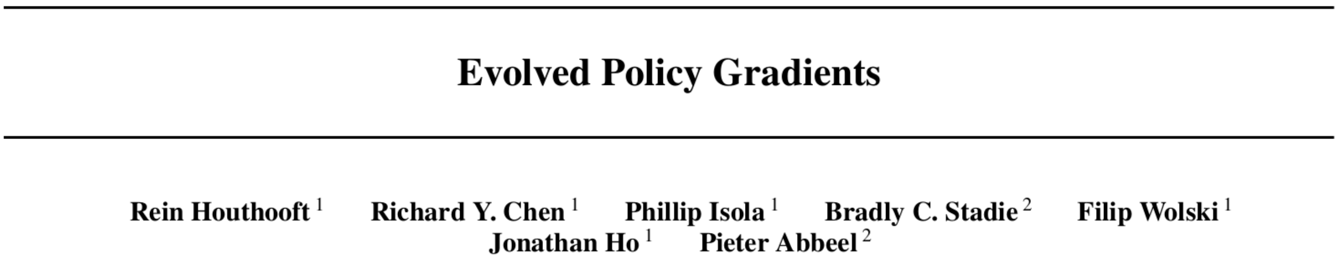
1802.04821 - Evolved Policy Gradients



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]

- **Yunqiu Xu**
- Brief introduction
  - Pieter Abbeel 组关于 meta-learning 的新作品, 用于基于梯度的强化学习
  - 主要思想:
    - Envolve a differentiable loss function, which is parameterized via temporal convolutions over agent's experience
    - 基于meta-learning学一个loss function
  - 这里引入的时序卷积在DAML中也被提及, 在DAML中:
    - 时序卷积被用于构造adaptation objective L_ψ, 应该也是一种 loss function 吧
    - The parameters are updated via $\phi = \theta - \alpha \nabla_\theta L_\psi(\theta, d^h)$
    - Further reading: 1802.01557 - One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning
  - Preformance:
    - Learns faster on several randomized environments
    - During testing, optimized only learned loss function, **does not require explicit reward signal**

1. Introduction

- Challenge:
  - Sometimes we can not obtain explicit internal rewards but simple feedback from teacher (1706.03741 - Deep Reinforcement Learning from Human Preferences)
  - In the real world, there's no teacher

- Insight:
    - Get prior knowledge over tasks to make learning faster on a new task
    - How to encode knowledge: **through envolved loss function**
- Our method:
    - Meta learning:
        - 内循环: 选取一个任务并优化外循环提供的loss function
        - 外循环: 在经过内循环之后, 优化loss function的参数, 使其能够最大化给定object的回报
    - Why envolved policy gradients
        - 虽然内循环可用SGD优化, 但优化外循环较困难 $\rightarrow$ outer objective无法被显式表示为loss parameters的函数
        - 因此我们引入进化策略 (blackbox optimizer), 将 evolved loss L 看作参数化的 PG $\frac{\partial_L}{\partial_\pi}$
    - Advantages to introduce learned loss function
        - Encode prior knowledge
        - Optimize the true objective rather than short-term returns (which will lead to local optima)
- Our contribution:
    - 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

2. Related Work

- RL
    - MDP: 这里引用的Sutton的教材

- PG:
  $$L_{pg} = E[R_\tau log\pi(\tau)] = E\left[R_\tau \sum_{t=0}^{H} log\pi(a_t|s_t)\right]$$
- AC: change L_{pg} as
  $$L_{ac} = E\left[\sum_{t=0}^{H} A(s_t)log\pi(a_t|s_t)\right]$$
- Our method: **does not use hand-defind function such as L_{ac}, learns a loss function instead**

* MAML: 和我们工作相比, 原始的MAML框架限制过多, 且要求目标函数可微
* DAML:
  - MAML及MIL的延伸, 和本工作类似也引入了temporal convolutions
  - 不同:
    - 他们工作重在解决behavioral cloning存在的问题(无法使用人类视频指导机器), 需要demonstration, 而我们的工作不需要
    - DAML的目标函数同样受限于可微, 其内循环仅有一步SGD, 而我们的目标函数是long-horizon且不可微的, 因此我们的内循环可以运行更多timesteps
  - **此处存疑: 本工作的summary中提到通过meta-learning学习到了一个可微的loss function, 且基于大量环境动作序列, 这不就是用了demonstration么**
* 另外两篇工作和我们工作类似, 也在构建reward function上做文章
  - 1706.03741 - Deep Reinforcement Learning from Human Preferences: 使用人的feedback来加速学习过程, 并尝试解决一些难以定义reward function的问题
  - Inverse RL: 尝试从demonstration中反向推导出reward

3. Methodology

* 本工作学到的loss function包含基于以往任务的时序卷积, 该loss function一方面可以内化(internalizing)环境回报, 另一方面可以激励较好的行为(如exploration)
* ϕ: parameters of learned loss function
* π_θ: train agent's policy

3.1 Meta-learning Objective

- Inner loop:

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

- Outer loop:

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

3.2 Algorithm

- 一方面我们难以用L_ϕ的显式函数表示最终返回的R_τ, 另一方面 Eq.3. 无法用基于GD的方法直接来解, 因此我们用基于进化策略的如下算法对外循环的loss function进行优化

Algorithm 1: Evolved Policy Gradients (EPG)

```
Input: Learning rates $\delta$ and $\alpha$, noise standard deviation $\sigma$, random environment distribution $p(\mathcal{M})$
[Outer Loop] for epoch $e = 1, \ldots, E$ do
    [Inner Loop] for each worker $i = 1, \ldots, n$ do
        Sample random vector $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathrm{I})$ and calculate the loss function parameter $\boldsymbol{\phi} + \sigma\boldsymbol{\epsilon}_i$
        Generate a random environment $\mathcal{M}_i$ according to $p(\mathcal{M})$
        for trajectory $j = 1, \ldots, K$ do
            Sample initial state $s_0$
            for timestep $t = 0, \ldots, M$ do
                Sample action $a_t$ from $\pi_{\boldsymbol{\theta}}(a_t|s_t)$
                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 $\boldsymbol{\theta}$ based on the loss function $L_{\boldsymbol{\phi}+\sigma\boldsymbol{\epsilon}_i}$ according to Eq. (4)
        Compute the final return $R_i$
    Update the parameter $\boldsymbol{\phi}$ for the loss function $L_{\boldsymbol{\phi}}$ according to Eq. (5)
```

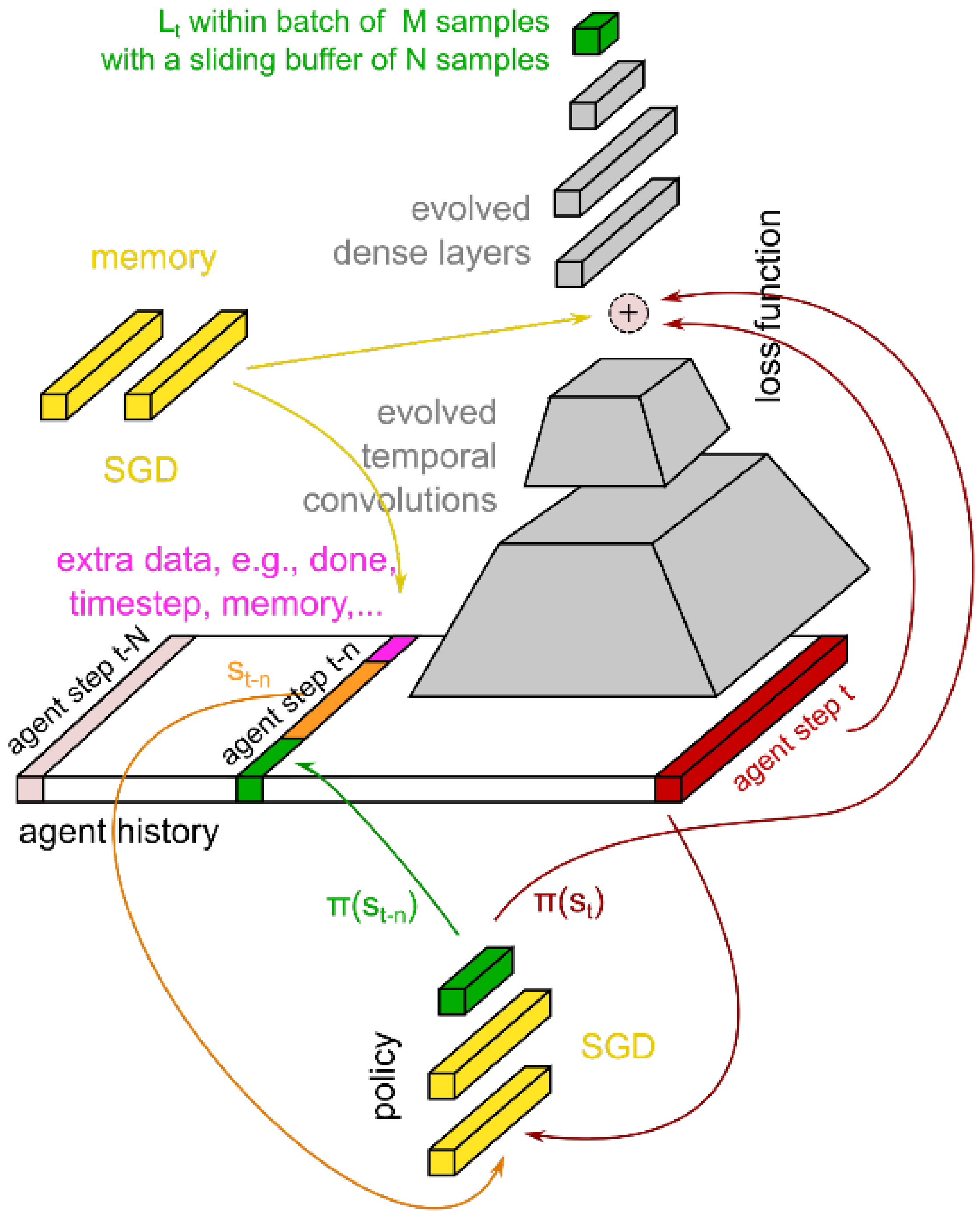
- 内循环: 在每个trajectory结束后, 使用GD来优化loss function L_i, 并将返回值 R_i 传递给外循环

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

- 外循环: 根据所有的内循环返回值 $\{R_i\}_{i=1}^{n}$ 来更新 ϕ

$$\phi \leftarrow \phi + \alpha \cdot \frac{1}{n\sigma} \sum_{i=1}^{n} R_i \boldsymbol{\epsilon}_i, \tag{5}$$

3.3 Architecture



L_t within batch of M samples
with a sliding buffer of N samples
evolved dense layers
memory
loss function
SGD
evolved temporal convolutions
extra data, e.g., done, timestep, memory,...
agent step t-N
s_{t-n}
agent step t-n
agent step t
agent history
$\pi(s_{t-n})$
$\pi(s_t)$
policy
SGD

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.

REINFORCE (44) or PPO (30) surrogate loss function L_{pg}, making the total loss

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

4. Experiment

- 基于MuJoCo的随机连续控制环境Hopper以及Reacher

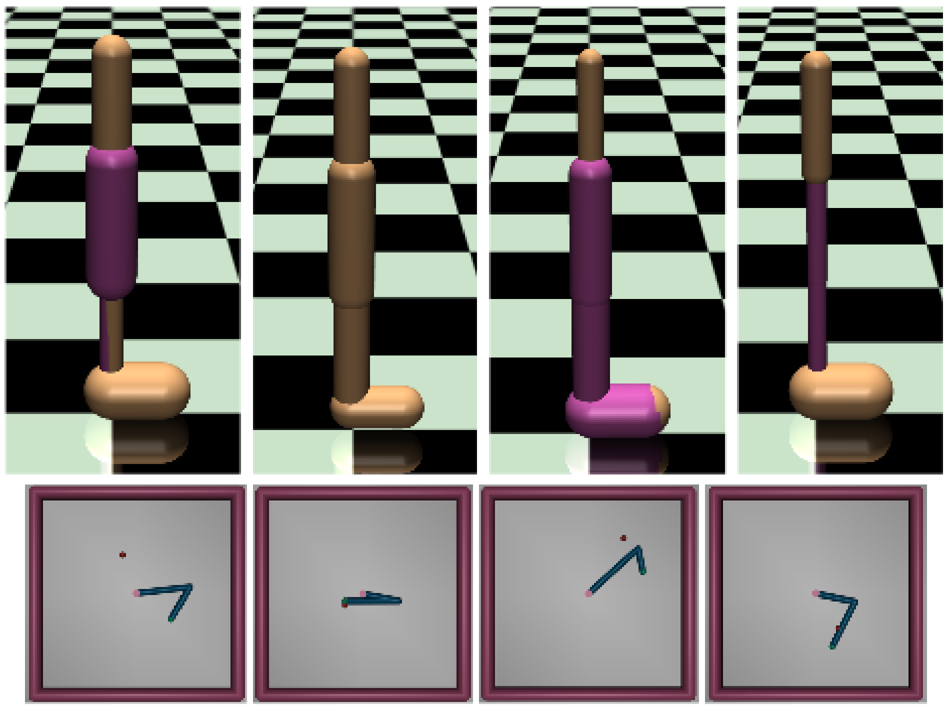


Figure 3: Examples of randomized MuJoCo environments. Top: Hopper with random limb thicknesses. Bottom: Reacher with random limb lengths.

- 本工作与PPO进行性能比较

4.1 Is the learned loss effective

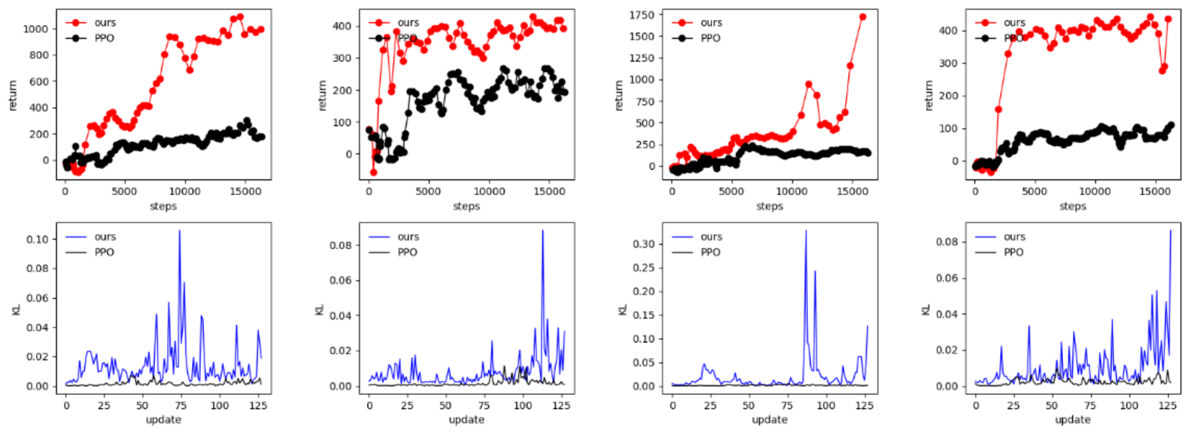


Figure 4: Random Hopper environment learning over 128 (policy updates) ×128 (timesteps per update) = 16384 total timesteps: PPO (black) vs no-reward evolved policy gradient algorithm (red). Each subplot column corresponds to a different randomization of the Random Hopper environment. These plots show 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.

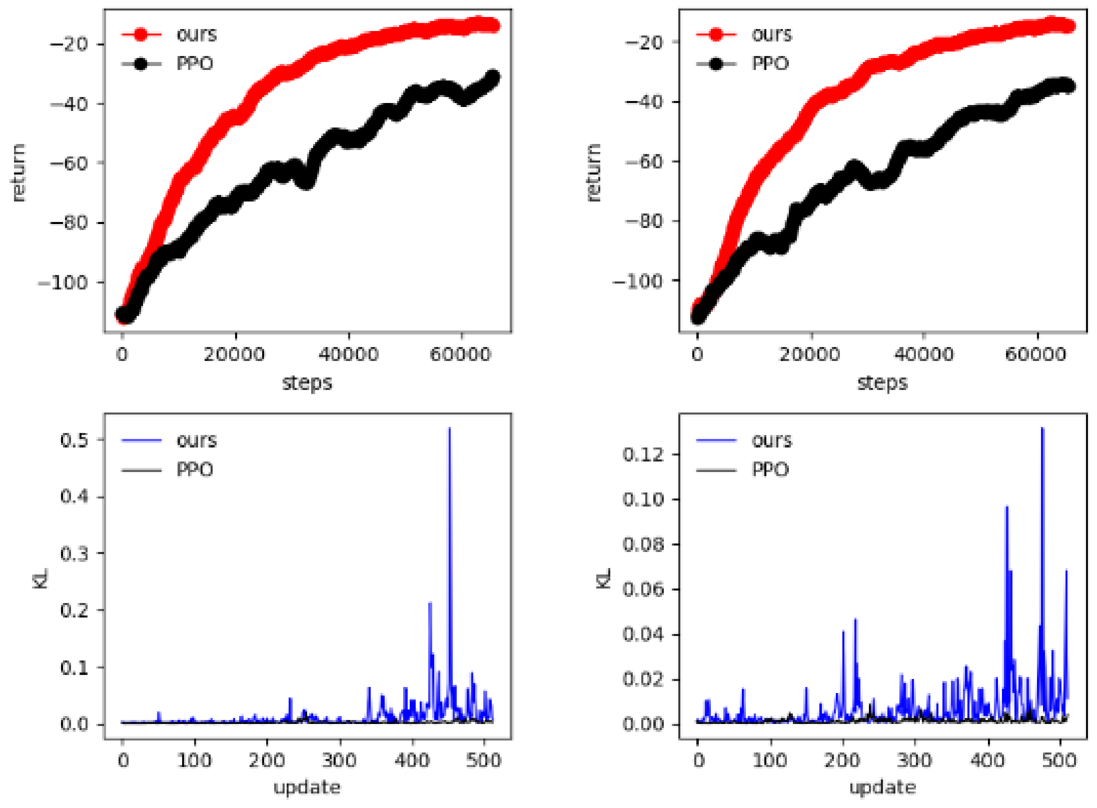


Figure 5: Random Reacher environment learning over 512 (policy updates) $\times 128$ (timesteps per update) $= 65536$ total timesteps: PPO (black) vs no reward evolved policy gradient algorithm (red). The two subplots correspond to two a different randomizations of the Random Reacher environment. These 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.

4.2 Does the loss encourage smooth learning

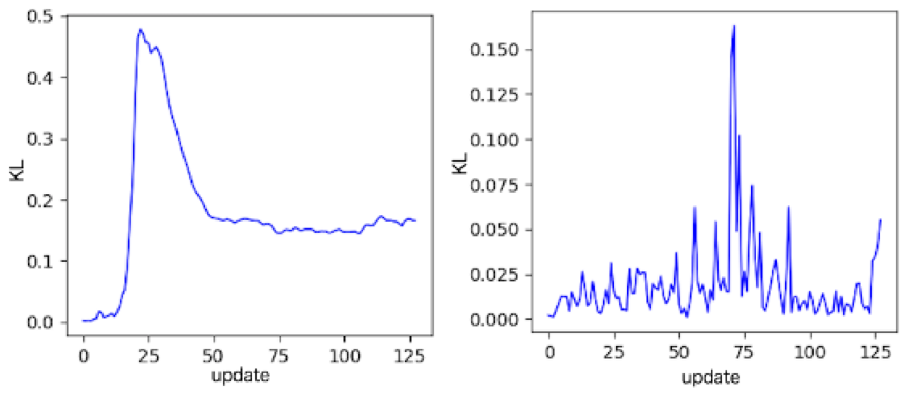
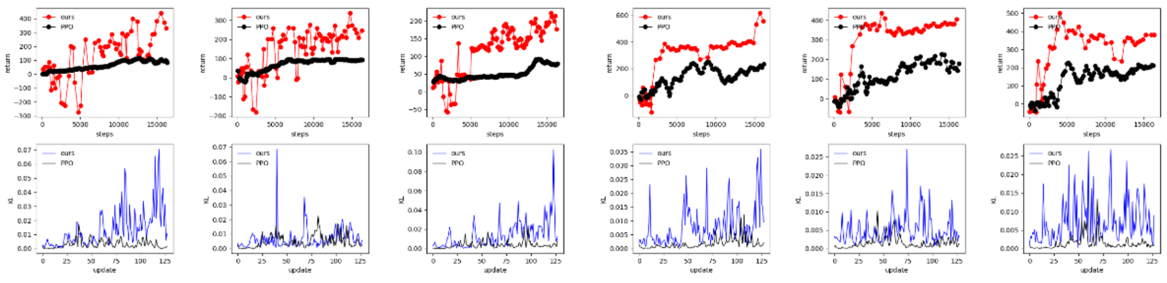


Figure 6: Evolving learning behavior in Random Hopper: the KL-divergence between adjacent policy updates in outer-loop iteration 1 (left) vs the last outer-loop iteration of meta-learning (without PPO guidance signal) (right).

4.3 Does the loss encourage exploration

- Forward-backward Hopper environment: 每个环境或者给予向前跳跃reward, 或者给向后跳跃
- Learning curve: learned loss is able to train agents that exhibit exploratory behavior



(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.

- The qualitative behavior of agent during learning

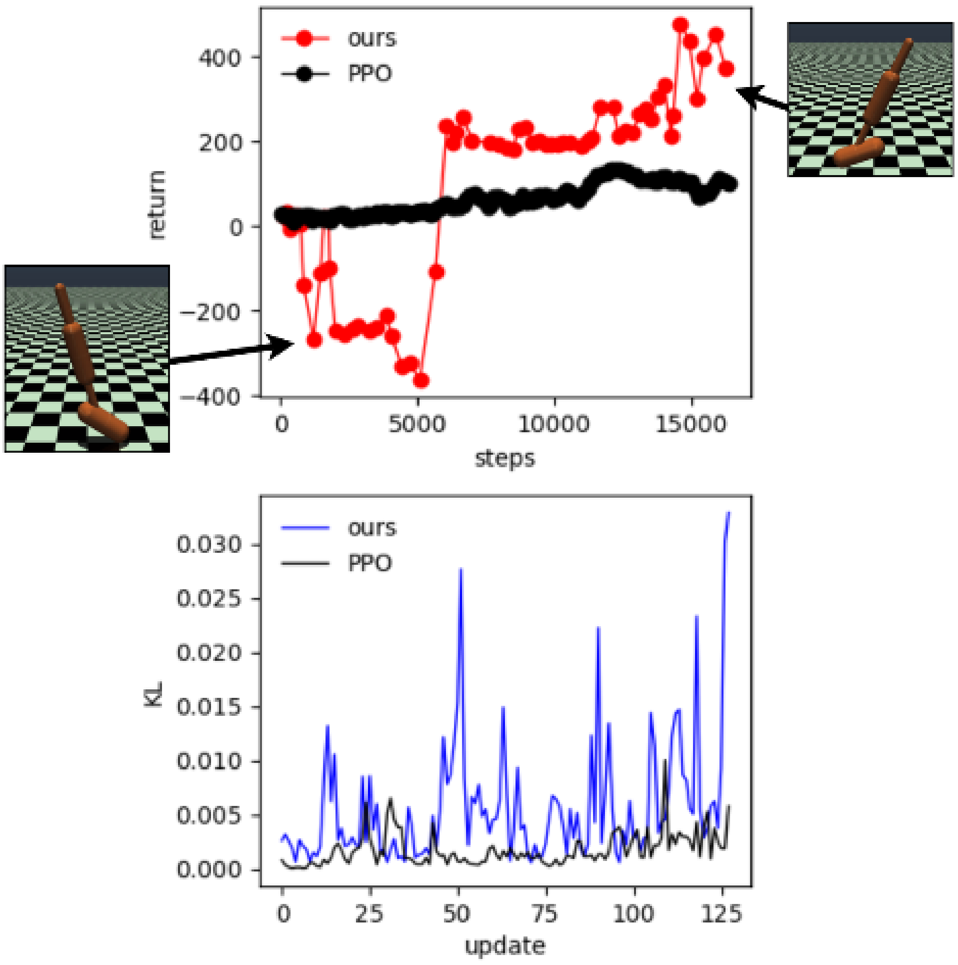


Figure 8: Forward-backward Hopper environment: exploratory behavior of evolved policy gradient algorithm (red) in figuring out the physics settings in comparison to PPO (black). The agent goes backwards for a while, then going forward and exploiting. Inlays indicate qualitative behavior observed in these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

4.4 How sensitive is the loss to different kinds of inputs

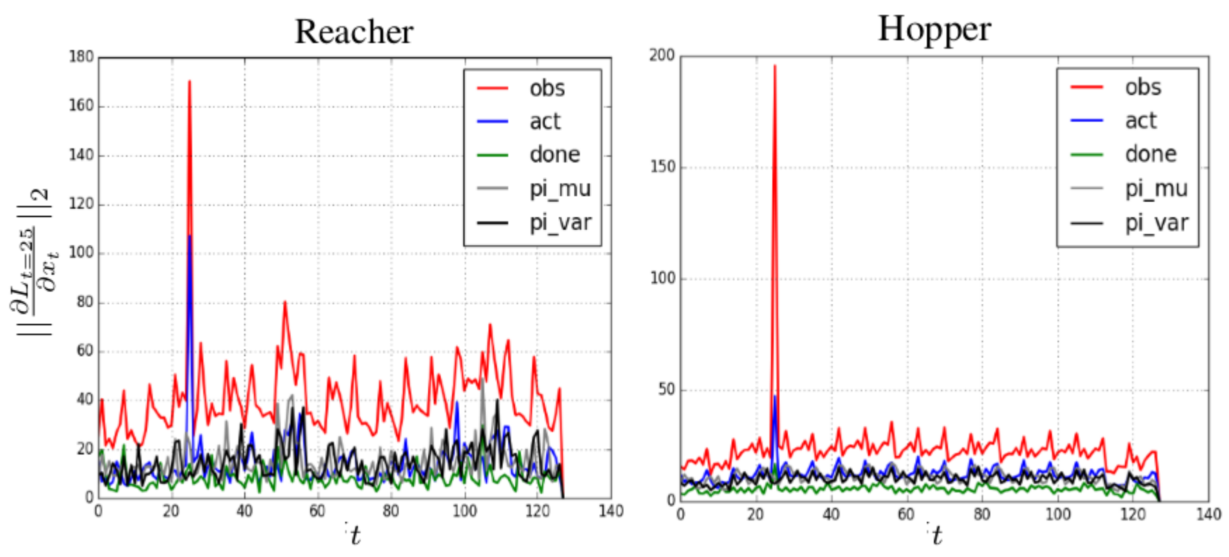


Figure 9: Which inputs is the learned loss function sensitive to? Here we plot the magnitude of the gradient of $L_{t=25}$ w.r.t. different kinds of inputs to the loss function at different time points within the input buffer. Notice that the loss depends highly on the data for the current time point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

5. Summary

- 本文工作:
  - 基于meta-learning, 从大量环境动作序列中学到可微的loss function
  - 和手动设计相比, 该loss function是adaptive(快速学习新任务)以及instructive的(在测试时不需要环境提供reward, 例如其中一个实验我们不能仅仅根据观察值得到reward)
  - Why instructive: 该性质可被理解为loss function对训练过程中见过的reward结构的内化, 我的理解是, **对于一个新任务, 如果以前见过类似任务, 便可以大致猜出这个新任务的reward**
- 未来工作:
  - 本工作需要序列化学习(在更新i+1前需要先通过外循环更新i), 在未来工作中可以进行并行化来提升效率
  - 效率提升后, 可尝试更有挑战性的任务