Replicating Deep RL from "Human" Preferences in Robotics Tasks

Lora Xie [1]

[1]Stanford University

Introduction

This project attempts to replicate some of the Gym robotics experiments in the seminal paper Deep Reinforcement Learning from Human Preferences[1]. A common bottleneck with reinforcement learning is the difficulty of hand-crafting rewards for agent behaviors, especially when the environment and/or the task is complex and subtle, or when humans are only good at recognizing good behavior but not describing it. The original paper addresses these difficulties by learning a reward function from human preferences over segments of trajectories generated by the agent, which humans can often provide without precisely understanding the underlying reward dynamics, and which are easier for humans to maintain consistency across timesteps compared to absolute numerical scores. RLHF is a promising direction for training larger models that are more aligned with true human goals and values. In this project, I try to implement a minimal (small, clean, and educational) version of the architecture described in the paper, as inspired by the minGPT project [2], and briefly explore how various modifications affect the performance of the method.

Key Concepts

Preference

We say that preferences $\succ$ are generated by a reward function [1]$r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ if

$$\left(\left(o_0^1, a_0^1\right), \ldots, \left(o_{k-1}^1, a_{k-1}^1\right)\right) \succ \left(\left(o_0^2, a_0^2\right), \ldots, \left(o_{k-1}^2, a_{k-1}^2\right)\right)$$

whenever

$$r\left(o_0^1, a_0^1\right) + \cdots + r\left(o_{k-1}^1, a_{k-1}^1\right) > r\left(o_0^2, a_0^2\right) + \cdots + r\left(o_{k-1}^2, a_{k-1}^2\right).$$

The human judgments are recorded in a database $\mathcal{D}$ of triples $\left(\sigma^1, \sigma^2, \mu\right)$, where σ^1 and σ^2 are the two segments and μ is a distribution over $\{1, 2\}$ indicating which segment the user preferred.

Preference	$\mu(1)$	$\mu(2)$
σ^1	1	0
σ^2	0	1
Equal	0.5	0.5
Incomparable	`Discard`	`Discard`

Estimated Reward Function

We can interpret a reward function estimate $\hat{r}$ as a preference-predictor if we view $\hat{r}$ as a latent factor explaining the human's judgments and assume that the human's probability of preferring a segment σ^i depends exponentially on the summed reward over the length of the clip:

$$\hat{P}\left[\sigma^1 \succ \sigma^2\right] = \frac{\exp \sum \hat{r}\left(o_t^1, a_t^1\right)}{\exp \sum \hat{r}\left(o_t^1, a_t^1\right) + \exp \sum \hat{r}\left(o_t^2, a_t^2\right)}.$$

We choose $\hat{r}$ to minimize the cross-entropy loss between these predictions and the actual human labels:

$$\operatorname{loss}(\hat{r}) = -\sum_{\left(\sigma^1, \sigma^2, \mu\right) \in \mathcal{D}} \mu(1) \log \hat{P}\left[\sigma^1 \succ \sigma^2\right] + \mu(2) \log \hat{P}\left[\sigma^2 \succ \sigma^1\right]$$

References

[1] Paul Christiano. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)* (2017). DOI: `https://doi.org/10.48550/arXiv.1706.03741`.

[2] Andrej Karpathy. *minGPT*. Accessed on March 16, 2023. URL: `https://github.com/karpathy/minGPT`.

[3] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`. URL: `http://arxiv.org/abs/1707.06347`.

Learning from Human Pipeline

Figure 1. Schematic illustration of the RLHF approach: the reward predictor is trained asynchronously from comparisons of trajectory segments, and the agent maximizes predicted reward. From the original paper[1].

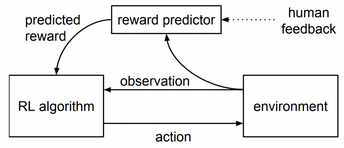
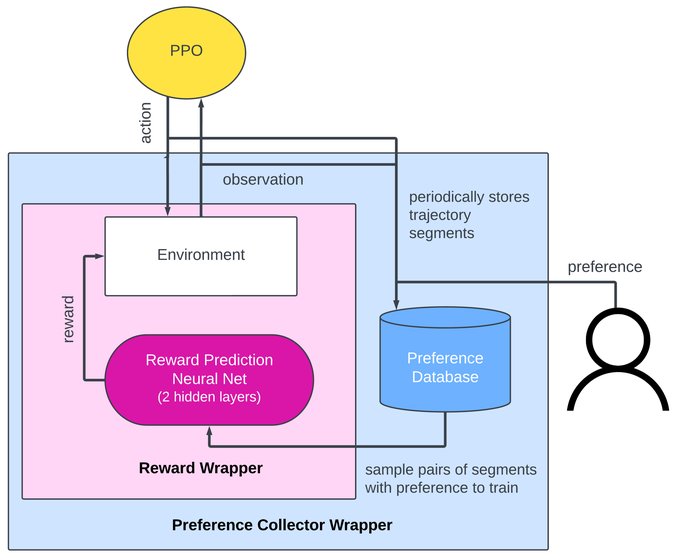


Figure 2. Our implementation of this interaction using environment wrappers.



- The learnable reward network and the preference collector are implemented as **wrappers** of the Gym environment. This way, the policy is completely independent from the RLHF structure. In this implementation, PPO[3] is used, but it can easily be switched out for any RL gradient policy. We can train the policy as normal, and the environment wrappers automatically handle simultaneous training of the reward network and the collection of preferences.
- Since human feedback is expensive to collect, in generating the performance comparisons in the next section, we used **synthetic feedback** as a proxy of human feedback: a synthetic oracle whose preferences over trajectories exactly reflect reward in the underlying task, i.e. it prefers whichever trajectory segment actually receives a higher reward according to the true reward function. According to the original paper, on most robotics tasks, using 700 synthetic queries leads to comparable reward increases as using 750 human queries.

Performance

We tested the system on the `Pendulum-v1` task in Gym to explore how different hyperparameters affect the agent's performance. All averaged results are over 3 runs. We trained 100 batches, 10000 steps per batch, and capped episode length at 1000.

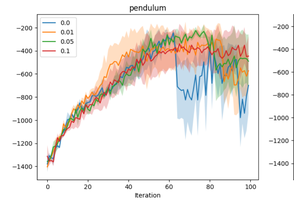


Figure 3. Entropy bonus added to PPO policy.

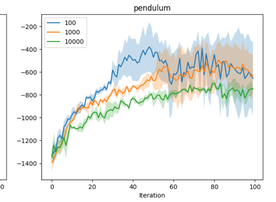


Figure 4. Number of steps passed per query.

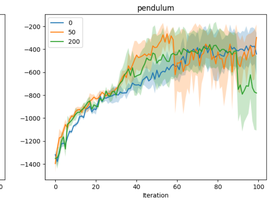
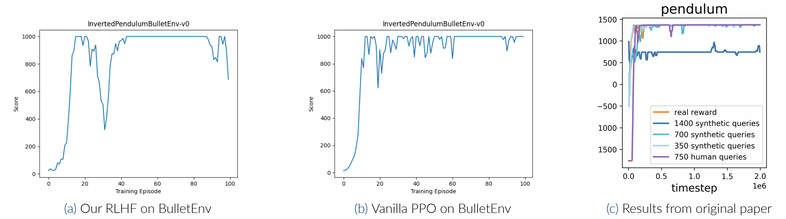


Figure 5. Number of queries before starting policy training.



(a) Our RLHF on BulletEnv (b) Vanilla PPO on BulletEnv (c) Results from original paper

Figure 6. Comparison with vanilla PPO and original paper

Analysis

- Entropy bonus provides critical stability. Without it, the policy almost always crashes after learning some desired behavior. The original paper used an entropy bonus of 0.01, but we found that the larger the entropy bonuses (up to 0.1) the better. This could be because we needed to make up for the omitted details in our simplified implementation (e.g. we did not use a decaying querying schedule, did not have an ensemble of reward predictors, etc.).
- A querying frequency of one per 1000 steps appears to be cost-efficient. Querying less frequently (every 10000 steps, 100 queries total) provides insufficient information to learn a good reward predictor. Querying more frequently does not lead to better performance, possibly because after about 1000 queries, the fitted reward function is already near optimal.
- Collecting some preferences before starting to train the policy helps boost early performance, but does not affect asymptotic behavior. Adding this feature could be valuable when computation cost is high and only a small number of epochs will be run.
- Compared to the vanilla PPO we implemented in assignment 3 (on `InvertedPendulumBulletEnv-v0`, a simpler environment to learn), our RLHF algorithm learns more slowly and is less stable, as expected, but eventually reaches the same optimal performance.