Replicating Deep Reinforcement Learning From Human Preferences in Robotic Tasks

Lora Xie [1]

Abstract

This project attempts to replicate a simulated robotics task experiment in the seminal paper Deep Reinforcement Learning from Human Preferences (Christiano, 2017). The original paper introduces a reinforcement learning (RL) method that defines goals in terms of (non-expert) human preferences between pairs of trajectory segments. The paper shows that a relatively small amount of online human feedback is sufficient for the agent to learn desired behaviors without access to the reward function. In this replication project, I implement a minimal (simplified, clean, and educational) version of the architecture described in the paper, as inspired by the minGPT project (Karpathy). I also briefly explore how various modifications affect performance. Source code of this implementation can be found here.

1. Background

The original paper[1] is motivated by a common bottleneck in reinforcement learning: the difficulty of hand-crafting rewards for agent behaviors. Often, the environment and/or the task is multifaceted, or complex, or subtle, or humans are only good at recognizing good behavior but not describing it. Some example tasks given in the original paper are cleaning a table and scrambling an egg, which despite being everyday tasks, are difficult to capture with a simple reward function that takes in as input the robot's sensors and that does not result in behaviors that optimize the reward function without actually achieving the intended goal. The paper addresses these difficulties by learning a reward function from human preferences over segments of trajectories generated by the agent. Humans can often provide such preferences without precisely understanding the underlying reward dynamics, and it is easier for humans to maintain consistency across time steps for preferences compared to absolute numerical scores.

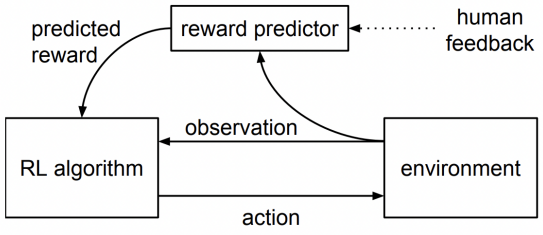

Figure 1. Schematic illustration of the RLHF approach: the reward predictor is trained asynchronously from comparisons of trajectory segments, and the agent maximizes predicted reward. Graph from the original paper.

2. Method Description

At each point in time our method maintains a policy $\pi : \mathcal{O} \rightarrow \mathcal{A}$ and a reward function estimate $\hat{r} : \mathcal{O} \times \mathcal{A} \rightarrow R$, each parameterized by deep neural networks. There are three main processes:

1. The policy π interacts with the environment to produce a set of trajectories $\left\{\tau^1, \ldots, \tau^i\right\}$. The parameters of π are updated by a traditional reinforcement learning algorithm, in order to maximize the sum of the predicted rewards $r_t = \hat{r}\left(o_t, a_t\right)$.

2. Pairs of segments $\left(\sigma^1, \sigma^2\right)$ from the trajectories $\left\{\tau^1, \ldots, \tau^i\right\}$ produced in step 1 are selected and sent to a human for comparison.

3. The parameters of the reward network $\hat{r}$ are optimized via supervised learning to fit the preferences collected from the human so far.

In order to train the reward network without access to the ground truth for the numerical reward of each step, the paper

[1]Compputer Science, Stanford University, Stanford, California, USA. Correspondence to: Lora Xie <loraxie@stanford.edu>.

CS234: Reinforcement Learning Winter 2023, Stanford, USA.

[1]Since this is a replication project, it inevitably makes a lot of references to this paper. For brevity, when it is clear that a statement comes from this paper, I will omit the parenthetical citation.

makes an assumption about how the collected preferences (over segments) relate to the underlying rewards. They view $\hat{r}$ as a latent factor explaining the human's judgments and further assume that the human's probability of preferring a segment σ^i depends exponentially on the summed reward over the length of the clip:

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

Then a meaningful cross-entropy loss can be defined between the predicted preferences and the actual human preferences: $\text{loss}(\hat{r}) = -\sum_{(\sigma^1,\sigma^2,\mu)\in\mathcal{D}} \mu(1) \log \hat{P}\left[\sigma^1 \succ \sigma^2\right] + \mu(2) \log \hat{P}\left[\sigma^2 \succ \sigma^1\right]$ where the database $\mathcal{D}$ contains triples $\left(\sigma^1, \sigma^2, \mu\right)$, with σ^1 and σ^2 being the two segments and μ being a distribution over $\{1, 2\}$ indicating which segment the user preferred, defined as follows:

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

3. Implementation Details

3.1. Simplification

Given the goal of implementing a minimal pipeline, we make the following simplification compared to the original paper, which is expected to impair performance to some extent:

1. Instead of training an ensemble of reward predictors, we train only one predictor.
2. As a result of having a single reward predictor, we cannot follow the original paper in prioritizing asking about segments for which there is the most disagreement among the ensemble members. The original ablation study using no ensemble picks queries uniformly at random. We further simplify this to asking for preference at a constant rate defined in the number of steps.
3. A constant querying rate is also a simplification of the decaying query schedule in the paper, where the rate of labeling after T frames is $2 * 10^6 / \left(T + 2 * 10^6\right)$.
4. We omit ℓ_2 regularization or dropout on the reward predictors, which is used in the original paper to keep the validation loss between 1.1 and 1.5 times the training loss.

There are two details that we do preserve: an entropy bonus for the policy, which will be further discussed, and normalizing the rewards produced by $\hat{r}$ to have zero mean and constant standard deviation, here implemented by batch norm. The latter is typical preprocessing, but it is particularly appropriate here since the cross entropy loss between predicted and actual preferences can teach only the ordering but not the absolute value of the rewards.

3.2. Pipeline

The learnable reward network and the preference collector are implemented as wrappers of the Gym environment (Brockman et al., 2016). This way, the policy is completely independent from the RLHF structure. The original paper used Trust Region Policy Optimization (TRPO) (Schulman et al., 2015), and in this implementation, the more modern Proximal Policy Optimization (PPO) (Schulman et al., 2017) is used, but it can easily be switched out for any RL gradient policy. In this pipeline, we train the policy as normal, and the environment wrappers automatically handle simultaneous training of the reward network and the collection of preferences.

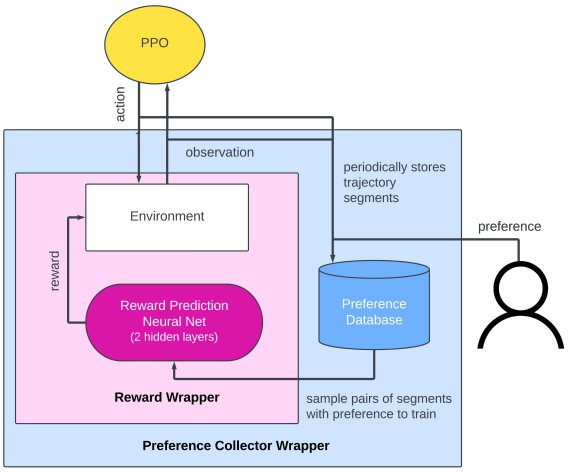

Figure 2. Pipeline for training a PPO agent from human preferences.

3.3. Synthetic Preferences

Since human feedback is expensive to collect, in generating the performance comparisons in the next section, as a proxy for human feedback, we use synthetic feedback generated by an oracle whose preferences over trajectories exactly reflect reward in the underlying task, i.e. it prefers whichever trajectory segment that actually receives a higher reward according to the true reward function. As observed in the original paper, on most robotics tasks, using 700 synthetic queries leads to comparable reward increases as using 750 human queries, and synthetic feedback is generally slightly better than human feedback. This makes synthetic feedback

a reasonably good approximation and means our following results are likely a slight overestimation of actual performance with real human preferences.

The source code for this project does include a human feedback version with an interface for actual human feedback. It displays videos of the pair of segments and collects preferences from keyboard input.

4. Results and Analysis

We test our implementation on the `Pendulum-v1` task in Gym. The pendulum task involves an arm attached at one end to a fixed point, and the goal is to apply torque on the free end of the arm to swing it into an upright position, given a random initial position.

We explore how different hyperparameters affect the agent's performance. We planned to test entropy bonuses, collecting preferences before training the policy, and different frequencies of asking for preferences. We ended up discovering two other important hyperparameters: the frequency of updating the reward network and the length of the segments to be compared.

All averaged results are over 3 runs. We train 100 batches, 10000 steps per batch, and cap episode length at 1000. The reward for any state-action pair (i.e. any step) is non-positive, ranging from -16.27 to 0. The scores (vertical axes) in the graphs below are the total reward over 200 steps. By default we ask for preference over a pair of segments every 1000 steps.

4.1. Entropy

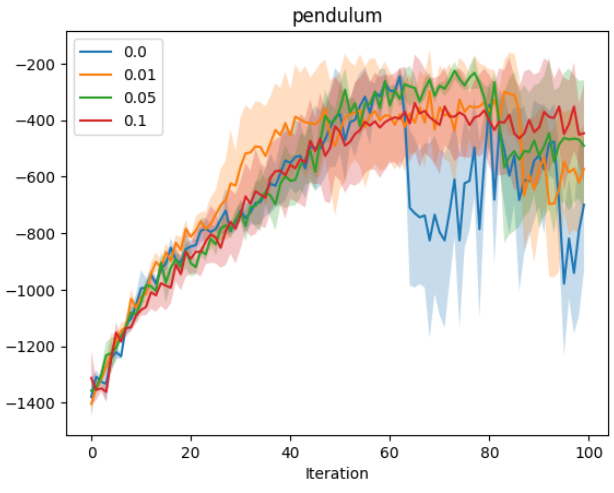

Figure 3. Adding different entropy bonuses to PPO.

An entropy bonus encourages exploration of unvisited states (Haarnoja et al., 2018). It is calculated from the entropy of the actions in a batch and is added to the predicted reward when training the policy. The theoretical explanation provided in the original paper for why entropy bonus is especially important for RLHF is that TRPO (and similarly PPO) "relies on the trust region to ensure adequate exploration, which can lead to inadequate exploration if the reward function is changing." Adding an entropy bonus is the only hyperparameter adjusted from a vanilla PPO policy itself (all others concern the RLHF pipeline).

In practice, without an entropy bonus, what we observe is that either the policy's scores never go past around -800 (visually this corresponds to the arm never swinging up high enough), or if the policy does learn some desired behaviors halfway through, in the not unlikely event that it makes some unfortunate exploration, it's difficult for the agent to recover from the crash, for given the current bad policy, it does not sufficiently explore radically different actions (the same reason that it sometimes barely makes any progress since the beginning).

The original paper uses an entropy bonus of 0.01, but we find that the larger the entropy bonus (up to 0.1) the better. This could be because our simplified implementation uses more aggressive exploration to make up for lost performance and stability due to the omitted details in 3.1.

4.2. Initial Preferences and Querying Frequency

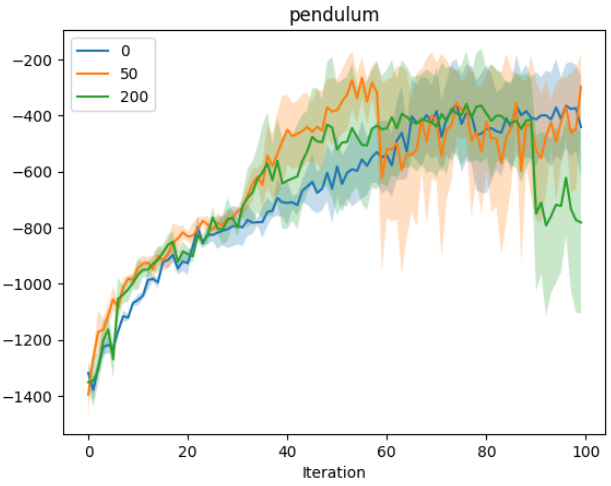

Figure 4. Number of queries before starting policy training.

The original paper collects 25% of their comparisons from a randomly initialized policy network at the beginning of training. We test having 0, 50, or 200 queries before starting to train the policy, in addition to the 1000 preferences collected online. We found that this add-on helps boost early perfor-

mance, but does not affect asymptotic behavior. This might be because given adequate exploration (thanks to entropy bonus) the reward for the pendulum task is easy to learn, so it is manageable to learn it along the way. For cases where compute is more limited and/or where the goal is more complex (e.g. humanoid tasks), adding initial queries might be more notably helpful.

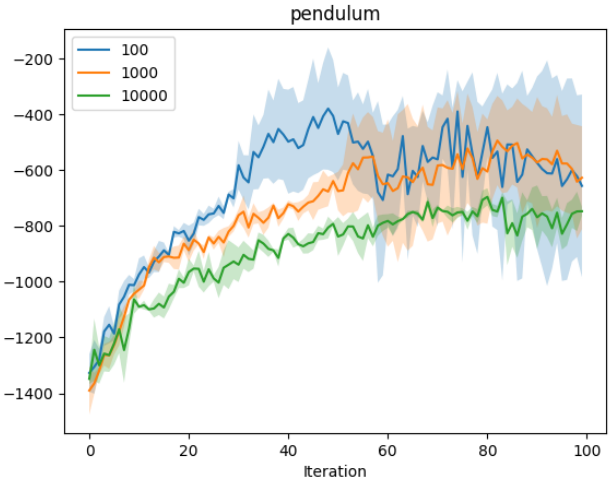

Figure 5. Number of steps per query.

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.

The fact that neither adding initial queries or increasing querying frequency past $\frac{1}{1000}$ further improves performance seems to indicate that with just 1000 online queries, we have a good estimate of the reward function, at least for this simple pendulum task at hand. However, across all three runs with different seeds, the performance of the policy plateaus after about 50 iterations, which seems like a sign that the learned reward network is not providing good enough reward signals, especially not sufficiently fine-grained rewards that can distinguish very good policies from somewhat good policies. To explore this, we investigate more explicitly the behavior of the reward network.

4.3. Reward Network Behavior

4.3.1. Spearman Correlation

Intuitively, learning the reward for each individual state-action pair in a segment requires a more exact estimate of the underlying reward dynamics than learning preferences over trajectory segments, which only requires getting the relative magnitude of the sums right. Thus, just because the reward network is achieving low training loss doesn't mean it is giving the policy the right reward values for each of its actions. On the other hand, it is also not reasonable to expect that the learned rewards are numerically similar to the true rewards, for the position of the rewards is underdetermined by our learning problem.

Thus we measure the Spearman correlation, a statistical measure of the strength of a monotonic relationship between the predicted and the true rewards, ranging from 0 to 1. What we see is that the correlation increases very quickly in the first 200000 steps or so, and then slowly decreases to as low as around 0. This pattern made us suspect overfitting, so we looked into a hyperparameter that had previously been neglected, namely the frequency at which the reward network was updated.

4.3.2. Reward Network Overfitting

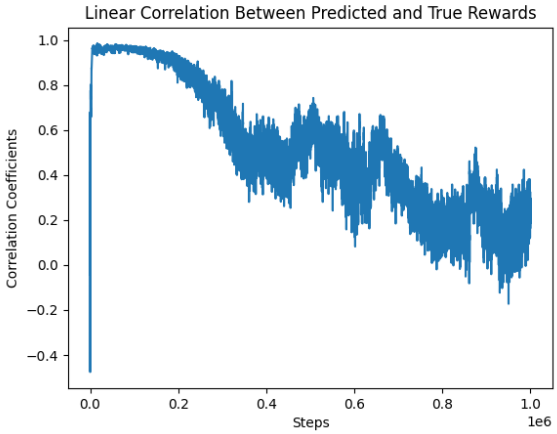

Figure 6. Pearson correlation between predicted and true rewards with reward network updated every 100 steps.

The previous results use a frequency of 1 per 100 steps, which means given our batch size of 64, on average the same pair of segments with their corresponding preference is used $\frac{1000}{10} \times 64 = 640$ times. It turns out that this is way too much. Lowering the reward network update frequency to 1 per 1000 steps significantly improves the correlation (above 0.5, which suggests moderate-strength monotonic relationship) later in training. This improvement is strong evidence that previously our reward network overfit its training data after training for too long.

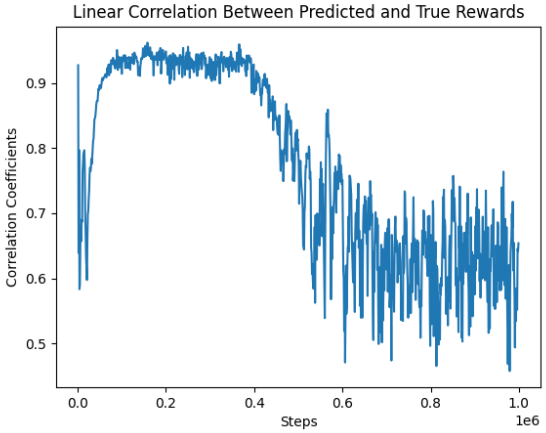

Figure 7. Pearson correlation between predicted and true rewards with reward network updated every 1000 instead of 100 steps.

4.3.3. Segment Length

Apart from overfitting, another hypothesis is that the segment length (previously set at 50 frames) might be too long. In the beginning, the policy's actions are more random. Visually this corresponds to the pendulum arm starting from various angles, swinging with various torques, and making relatively large movements, so 50 frames roughly allow the arm to go up once and fall down once (or vice versa) unless the arm is barely moving at the bottom. Thus each clip is more likely to be a coherent attempt. Given such a pair, it is likely that one clip is clearly better than the other; for example, maybe in one clip the arm swings over the top but not in the other.

However, as the policy learns better and better behaviors, it is going to stay at the top for longer, and while it is there it will make smaller movements. All of its movement in 50 frames might just be tiny balancing adjustments. In practice, by trajectory rewards around -200 (which is on average -1 per step), we observe that at any given point in time, the arm forms a $< 5°$ angle with the vertical line – visually very close to the optimal solution. Because each attempt (here we think of attempts as adjacent sequences of actions in the same direction) is so small, a lot of mostly independent attempts can fit in a 50-frame clip, so the preference based on all the rewards lumped together is going to contain more noise than information for each of the individual attempts.

In our experiment, reducing the segment length to 25 frames leads to a significant improvement in the correlation even towards the end of training. This observation confirms our hypothesis above.

Note that when the segment length is smaller, synthetic feedback might be significantly better than human feedback and thus no longer serve as a good proxy, because synthetic feedback is more sensitive and precise about angles and velocities, whereas humans can only eyeball and cannot reliably tell, for example, a $10°$ difference. Thus, there is a limit to how much we can exploit a shorter clip length.

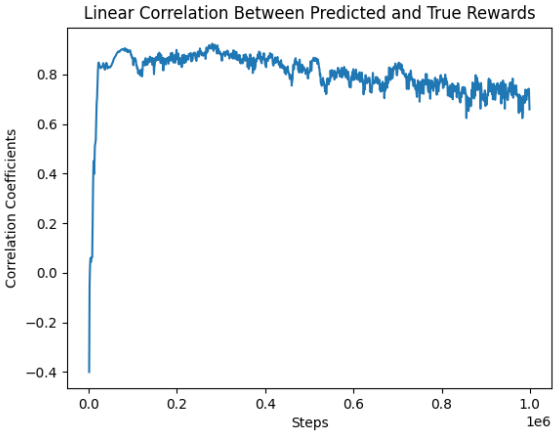

Figure 8. Pearson correlation between predicted and true rewards when each segment is 25 instead of 50 steps.

4.4. Comparison with RL from True Rewards

We compare our RLHF algorithm with the vanilla PPO with true rewards implemented in assignment 3 of CS234 Winter 2023 at Stanford University, on `InvertedPendulumBulletEnv-v0`, which is the environment used for that assignment and which is a simpler environment to learn. Our algorithm takes more wall-clock time due to the querying, but takes roughly the same amount of training (in terms of the number of iterations and steps) to reach and stay at the same optimal performance. This might be less true for tasks that are more complex, since the iterations it takes to learn a good estimate of the underlying reward function will likely make RLHF more unstable and require more training.

4.5. Comparison with Original Paper

Our simplified implementation demonstrates similar trends on a simple task as the original paper, but the performance is weaker, likely because of the implementation details we omit and because there are probably more hyperparameters that we neglect but can be significantly more optimized, exemplified by the reward network update frequency and segment length in 4.3. A more detailed quantitative comparison is difficult because the original paper uses a different pendulum task (i.e. in Mujoco (Todorov et al., 2012)), which has a different true reward with a different range.

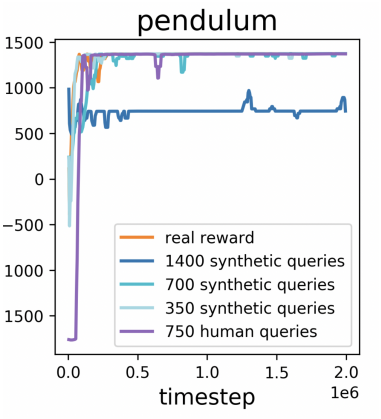

Figure 9. Pendulum results from the original paper.

5. Conclusion

Through this simplified implementation of deep reinforcement learning from human preferences and experiments on a relatively simple robotics task, we reach several findings that are likely applicable to RLHF problems more generally. First, using an entropy bonus to encourage exploration is helpful for adding stability in the face of a changing learned reward function. Secondly, for continuous actions with continuous observations, there is a limit to how precisely optimized an agent trained by RLHF can be, since there is a limit to the degree of precision in the feedback that humans can practically provide. Finally, because RLHF has a few components, there are more hyperparameters than usual that can affect the algorithm's performance and stability. Since the components are interactive, it might be a nontrivial task to find a set of hyperparameters that optimize all components.

Acknowledgements

I thank Professor Emma Brunskill for pointing me to the 2017 Christiano paper as a good replication project for RLHF, my mentor Skanda Vaidyanath for general feedback and for suggesting correlation between true and estimated rewards as a better measurement of reward network behavior than just preference prediction accuracy over trajectory segments, Dilip Arumugam for pointing out that entropy bonus is a critical element that cannot be omitted even in a simplified implementation, and Anirudhan Badrinath for help with capturing videos in Gym.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Christiano, P. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017. doi: https://doi.org/10.48550/arXiv.1706.03741.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

Karpathy, A. minGPT. URL https://github.com/karpathy/minGPT. Accessed on March 16, 2023.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.