INSTITUT POLYTECHNIQUE DE PARIS

Deep Reinforcement Learning from Human Preferences

Reinforcement Learning Project

Students: Pierre LOVITON
Angie MÉNDEZ-LLANOS

February 2023

Project

Contents

Abstract

Deep reinforcement learning is a promising area of research for developing intelligent agents that can learn to perform complex tasks through trial-and-error. However, training reinforcement learning agents can be challenging, as it typically requires a large amount of trial and error experience. To address this challenge, the paper 'Deep Reinforcement Learning From Human Preferences' introduces a new approach for training agents that leverages human feedback to guide the learning process. The authors propose a framework that combines preference-based feedback from humans with deep neural networks and inverse reinforcement learning to learn the underlying reward function of the task. The proposed method is demonstrated to outperform traditional reinforcement learning methods in terms of both task performance and sample efficiency. This approach has potential applications in fields such as autonomous driving, robotics, and gaming, where it may be challenging to obtain a large amount of trial-and-error experience.

1. Introduction

Our code and github :
https://github.com/lovitonp10/Reinforcement-Learning-Project-M2DS-X

Traditional Deep RL introduction :
Deep reinforcement learning is a type of machine learning that involves training artificial agents to make decisions in complex, uncertain environments. These agents are trained to maximize a reward signal by interacting with their environment and learning from the consequences of their actions. This is different from other types of machine learning, where the goal is to learn from a fixed dataset and make predictions about new data.

Deep reinforcement learning combines artificial neural networks with a framework of reinforcement learning to help software agents learn how to achieve their goals. This involves mapping states and actions to the rewards they lead to, using function approximation and target optimization. Recent AI breakthroughs in problems like computer vision and machine translation have been made possible by neural networks, and they can also be used in combination with reinforcement learning algorithms to create powerful AI systems like Deepmind's AlphaGo. Reinforcement learning algorithms are goal-oriented, learning to maximize a particular dimension over many steps. These algorithms can start from scratch and, under the right conditions, achieve superhuman performance. They are incentivized by rewards and punishments, a process known as reinforcement, to help them to make the right decisions.

Aside from the agent and the environment, a reinforcement learning model has four essential components: a policy, a reward, a value function, and an environment model.

Definition of policy :

A policy is the agent's internal strategy on picking actions. It is the fundamental part of reinforcement learning that allows agents to make the right choices while interacting with the environment. It defines the learning agent's way of behaving at a given time.

- Deterministic policies: maps state and action directly:

$$u_k = \pi(x_k)$$

- Stochastic policies: maps a probability of the action given a state:

$$\pi(U_k|X_k) = P[U_k|X_k]$$

- RL is all about changing π over time in order to maximize the expected return

Definition of reward :

In reinforcement learning, the reward serves as the goal for the agent to optimize. At each time step, the agent's actions result in a reward or punishment. The overall objective of the agent is to maximize the cumulative reward it receives. The reward is used to differentiate between good and bad outcomes for the agent, and helps to guide the learning process by providing feedback on the effectiveness of different actions. In natural systems, rewards and punishments are often associated with pleasurable or unpleasant experiences.

- A reward is a scalar random variable R_k with realizations r_k.
- All goals can be described by the maximization of the expected cumulative reward:

$$\max \mathbb{E}\left[\sum_{i=0}^{\infty} R_{k+i+1}\right]$$

Definition of value function :

The value of a state in reinforcement learning is the sum of all future rewards that an agent can expect to receive if it starts in that state. The value of a state reflects the long-term attractiveness of a group of states based on the expected future states and the rewards produced by those states. Even if a state provides a small immediate reward, it may still be valuable if it often leads to additional states with higher rewards.

Definition of environment model :

The environment model is another significant component of several reinforcement learning systems. This is a mechanism that mimics environmental behavior and enables predictions about how the environment will respond. This model will let the agent forecast the next reward if an action is taken, enabling the agent to rely on developing action on future environmental responses.

Deep RL from Human Preferences :
In the context of this paper , human preferences refer to the preferences, opinions, or desired outcomes expressed by human users. These preferences are used as a guiding signal to train a reinforcement learning (RL) algorithm, with the goal of creating an RL agent that makes decisions aligned with the human preferences. The human preferences can be collected through various methods such as interactive feedback, demonstration, or questionnaires.

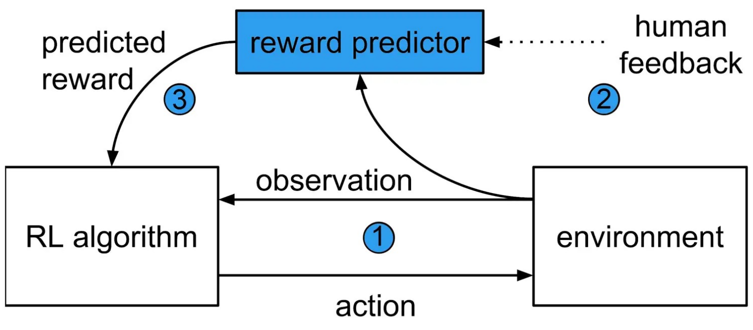

Figure 1: Reward predictor trained asynchonously on comparisons of trajectory segments

2. Previous Work

There are many papers in the literature that take into account human rankings or ratings in reinforcement learning settings. We give a brief description of a couple of papers that follow this line:

- **Preference-Based Policy Learning** [13]: The paper presents a direct policy learning method called Preference-based Policy Learning (PPL), which does not require a simulator. PPL follows a four-step process: the robot demonstrates a policy, the expert ranks it relative to other policies based on preferences, the preferences are used to estimate policy return, and the robot uses the return estimate to generate new candidate policies. This process is iterated until the desired behavior is achieved. It allows to teach relatively simple behaviors involving one robot, such as finding the way out of a labyrinth, or two, such as collaboratively exploring an arena.
- **Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning** [14]: This work presents a continuous actor-critic reinforcement learning method for optimizing the control of multi-function

myoelectric devices to create adaptable, intelligent artificial limbs. The method uses a simulated upper-arm robotic prosthesis and a sparse human-delivered training signal to derive successful limb controllers from myoelectric data without requiring detailed knowledge about the task domain. This framework can be used by both patients and clinical staff and can be easily adapted to different application domains and the needs of individual amputees.

- **Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning** [15]: The paper proposes a new model-free reinforcement learning algorithm called model-free PBRL that integrates preference-based estimation of the reward function. The algorithm is based on Relative Entropy Policy Search (REPS) and allows the use of stochastic policies and direct control of policy update greediness. The preference-based estimation is calculated using a sample-based Bayesian method that can estimate the uncertainty of the utility. The paper also compares the new algorithm to a linear solvable approximation based on inverse RL. REPS is used to limit the relative entropy of the policy update, which decreases exploration of the policy slowly and ensures that the algorithm is provided with informative preferences.

There are also papers in different domains were human preferences are optimized outside of reinforcement learning settings, we also give a brief description of some of these:

- **Picbreeder: evolving pictures collaboratively online** [16]: The paper describes an online service called Picbreeder, which allows users to evolve images collaboratively using an Interactive Evolutionary Computation (IEC) approach. Users select appealing images to produce a new generation of images, which can be shared with an online community. Picbreeder offers the ability to continue evolving other users' images, resulting in the proliferation of evolved images through the NeuroEvolution of Augmenting Topologies (NEAT) algorithm. This allows for the creation of more complex images than other IEC systems. The service is accessible to the entire internet community, and the paper showcases how Picbreeder encourages innovation by featuring collaboratively evolved images.
- **A robust activity marking system for exploring active neuronal ensembles** [17]: The paper introduces a new system called Robust Activity Marking (RAM) for identifying and studying ensembles of neurons. The system utilizes an optimized synthetic activity-regulated promoter that is strongly induced by neuronal activity and a modified Tet-Off system that achieves improved temporal control. The RAM system provides high sensitivity and selectivity, enabling

researchers to investigate neuronal activity with unprecedented precision.

The authors base their work in the same approach proposed in [18] and [3]. The key difference is that the domains and behaviors used are more complex, which makes it necessary to explore different Reinforcement Learning algorithms, reward models and, training strategies. One main difference between the paper we are studying and [18] and [3] is that, in those papers, instead of using short segments, they use full trajectories for human evaluation. The main consequence of this is that it requires more human time per data point. The idea of using short trajectories for human evaluation was inspired by [19]. However, in [19], the reward function works differently and they didn't get to test their algorithms with real human feedback.

In essence, the main contribution of this paper is to extend human feedback to deep reinforcement learning in order to teach more intricate behaviors. The authors point out that this is a new development in the area of scaling reward learning methods to large deep learning systems. They cite a number of examples of this trend, such as inverse RL, imitation learning, semi-supervised skill generalization, and bootstrapping RL from demonstrations. In comparison to previous research, this paper seeks to apply human feedback to more complex issues and demonstrate the efficacy of the technique in scaling up reinforcement learning.

3. Model description

In this section, we present the model and provide an overview of the key concepts necessary to understand the approach of the authors. We provide a brief overview of the reinforcement learning algorithms used in the experiments. In addition, we discuss how the authors incorporate human feedback into their approach, which is a crucial aspect of their work. By introducing and explaining these concepts, we provide a foundation to understand the subsequent sections, where we go into detail about about the experiments performed, the results gotten by the authors and the results of our own experiments.

3.1 Baseline and optimization of the policy

An agent interacts with the environment over time (a series of steps). At time t, the agent receives an observation $o_t \in \mathcal{O}$ that comes from the environment and then return an action $a_t \in \mathcal{A}$ (to the environment). In this case, instead of getting a reward from the environment, we assume that there is a human expressing preferences over trajectory segments, where a trajectory segment is a sequence of pairs $((o_t, a_t) \in \mathcal{O} \times \mathcal{A}$.

A trajectory segment then defined as follows:

$$\sigma((o_0, a_0), (o_1, a_1), \ldots, (o_{k-1}, a_{k-1})) \in (\mathcal{O} \times \mathcal{A})^k$$

We note $\sigma^1 \succ \sigma^2$ to indicate that the human chose σ^1 over σ^2 as a preferred trajectory. In simple terms, the objective of the agent is to produce trajectories preferred by the human while producing as little queries as possible for the human.

For each step time t, there exists a policy $\pi : \mathcal{O} \to \mathcal{A}$ and an estimate of the reward function $\hat{r} : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$.

1. The policy π produces trajectories $\{\pi^1, \ldots \tau^i\}$ by interacting with the environment. The parameters of π are updated at each step (via a traditional RL algorithm) so that the sum of the predicted awards $r_t = \hat{r}_t(o_t, a_t)$ is maximized.
2. Then a pair of segments (ω^1, ω^2) from the trajectories generated in step 1. These segments are then sent to a human to compare and choose the preferred one.
3. Then, supervised learning is used to optimize the parameters of the reward function $\hat{r}$ so they match the comparisons made by the human up to that point in time.

These three steps are followed in an asynchronous manner starting in (1) to go to (2) for human evaluation. Then these evaluations go to step (3) where $\hat{r}$ is optimized and its parameters are sent to step (1) to start over. We have a traditional reinforcement learning problem after using $\hat{r}$ to compute rewards. To preprocess, the reward $\hat{r}$ is standardized so it has zero mean and constant standard deviation. Afterwards, a suitable algorithm must be chosen to solve the RL problem. Since $\hat{r}$ may be non-stationary, meaning that it changes in time, it is better to go for algorithms that can handle better changes in the reward function. In this kind of problems, policy gradient methods have been successfully used. Policy gradient methods are a type of RL technique uses gradient descent to optimize parametrized policies with respect to the expected return. One algorithm following this method was chosen for each of the experiments.

3.1.1 Algorithms

We describe each of the algorithms used to solve the Reinforcement Learning problem in each experiment. The first algorithm is the *Advantage critic-actor*, used for the Atari games experiment. It involves the use of neural networks and Q-values. The second algorithm is *Trust region policy optimization*, used in the simulated robotics experiment, which parts from gradient descent and uses its advantages while dealing with its disadvantages.

- **Advantage actor-critic**

  As the name suggests, there are three main components to this algorithm: The advantage function, which calculates the advantage, TD error or prediction error. The actor network, which chooses an action. And finally, the critic network which evaluates the value of a state. The algorithm uses Temporal Difference Learning in which an agent learns by making predictions on future rewards and adjusting the actions based on the Prediction Error. This prediction error is calculated using the Advantage Function. The TD error is calculated as follows:

  $$\text{TD\_error} = \text{TD\_Target} - V(S) \tag{1}$$

  Where $V(S)$ is the value of the state S (it represents the critic network) and the TD target is computed at each step based on the reward R at the current state, the value of the discounted future rewards represented by γ and $V(S')$, where S' is the next state. In all steps, except the last one, the formula for the TD error is

  $$\text{TD\_Target} = R + \gamma * V(S')$$

  and for the last step, the formula is $\text{TD\_Target} = R$. This makes sense since at the last step there is no next state to calculate the the discounted future reward.

  The goal of the Advantage function is to decide if a given reward is better or worse than expected. For example, if an action leads to a negative advantage, we would like for the probability of that action to be smaller. Similarly when an action leads to a positive advantage.

  The actor network is a neural network that aims to map states to action probabilities. This is, given a state, the actor network gives an estimate of the probability of choosing certain action. The parameters of the neural network are trained based on the feedback given by the agent. If the agent decides that taking certain path is not the correct decision, then the weights are updated such that the probability of taking that path is smaller and the other probabilities increase in consequence. The agent makes this decision using the advantage function which can take positive or negative values to encourage or discourage certain action.

  The critic network is another neural network that maps the states to $Q-$values. The $Q-$value represents the values of a given state. The output of this neural network is the TD_target defined before, this means that the final layer of the neural network is a single neuron. The goal is for the TD_error to go to zero so the weights of the critic network are updated by optimizing the TD Error function using the Mean Square Error as loss function.

- **Trust region policy optimization**

  Trust region policy optimization aims to solve the problem of methods using gradient descent only for the optimization part. The problem with gradient descent is that it can be hard to handle the step size at every iteration. This step size can be too large or too small and one step size could be really good for one iteration, but completely inaccurate for another one, which means that it would be useful to have something that changes over time. One way to handle this is by including second-order derivatives in this is what Natural Policy Gradient methods are based on. In order to capture the sensitivity of changes in the step size, the KL-divergence is computed:

$$\mathcal{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta+\Delta\theta}) = \sum_{x\in\mathcal{X}} \pi_\theta(x) \log\left(\frac{\pi_\theta(x)}{\pi_{\theta+\Delta\theta}(x)}\right),$$

  where π_θ is the policy. In a nutshell, the KL-divergence measures how much the policy changes when there is a change in the parameters. To compute the optimal $\Delta\theta$, we use the KL-divergence to put a limit on how big this jump should be and we choose the one that maximizes the cost function. So the optimal $\Delta\theta$ is defined as follows:

$$\Delta\theta^* = \arg\max_{\mathcal{D}_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta+\Delta\theta}) \leq \epsilon} J(\theta + \Delta\theta).$$

  This definition allows to take the largest update step within a zone bounded by ϵ in terms of the divergence. Using the Taylor expansion to compute the gradient (on the objective) and the Hessian (on the KL-divergence), we get the following weight update formula:

$$\Delta\theta = \sqrt{\frac{2\epsilon}{\nabla J(\theta)^\top F(\theta)^{-1} \nabla J(\theta)}}\, \tilde{\nabla} J(\theta)$$

  where $F(\theta)$ is the Fisher information matrix of the objective function. Note that the step size is no longer fixed for all the iterations, it changes depending on the region where the point is in a given iteration. That means that in flat regions the step size can be large to converge faster towards the minimum whereas in curved regions it can be smaller to advance with caution. Note also that this method implies not only the computation and storage of a $\dim(\theta) \times \dim(\theta)$ matrix, which can lead to memory problems when there are a lot o parameters (often the case of neural networks), but also the computation of its inverse. The goal of TRPO is to use the advantages of second-order methods (compared to first-order methods like gradient descent) while dealing with the disadvantages of Natural Policy Gradients. Using the weight update formula from natural gradients, we derive the following equation to compute the Fisher matrix:

$$F(\theta) = \mathbb{E}_\theta[\nabla_\theta \log \pi_\theta(x) \nabla_\theta \log \pi\theta(x)^\top]$$

Recall that one of the sub-problems to solve is computing the inverse of the Fisher matrix. TRPO handles this by using the conjugate gradient method to approximate $x = F^-1\nabla \log \pi_\theta(x)$ which often converges within $|\theta|$ steps or even less. The natural gradient method is supposed to provide an optimal step size based on a constraint on KL-divergence, but in practice, approximations can prevent the constraint from being met. To solve this problem, TRPO uses a line search approach, which iteratively reduces the size of the update until the constraint is satisfied. This process is similar to shrinking the trust region, which is the area in which we believe the update will improve the objective. TRPO uses an exponentially decaying rate α^j to reduce the trust region, where α is a value between 0 and 1 and $j \in \mathbb{N}$. If the first update meets the constraint, the original natural gradient step size is preserved. If not, the trust region is reduced further until a satisfactory update is achieved. So, the main difference in this regard is that while natural policy assumes that the divergence constraint is satisfied, TRPO goes a step further by actively enforcing it. The final difference is quite logical. Instead of assuming that the update will enhance the surrogate advantage $\mathcal{L}(\theta)$, TRPO verifies it. To do this, we compute advantages based on the old policy, and then use importance sampling to adjust the probabilities. While this verification step takes some time, it's helpful to make sure that the policy is actually improving before accepting the update.

3.2 Human preference and reward function fitting

For the human evaluation, the authors produced short clips between 1 and 2 seconds long. These videos correspond to two different trajectories and the human reviewer could choose between three different options: choosing one of the two trajectories, deciding that the trajectories are the same or deciding that the trajectories are not comparable.
This is stored in a database $\mathcal{D}$ that contains a triple $(\sigma^1, \sigma^2, \mu)$ per choice. In each triple, σ^1 and σ^2 represent the two trajectories and μ is a distribution over $\{1, 2\}$ that describes the choice. If the human chose one of the two trajectories, then all the mass of μ goes to the corresponding σ^i. If the human stated that the two trajectories are the same, then μ is uniform. Finally, the cases were the human chose that the two trajectories are not comparable are not included in the database.

For the fitting, the authors assume an estimate of the reward, $\hat{r}$, to be a latent or 'hidden' factor and interpret it as a preference-predictor that explains the human preference. The probability of choosing one of the segments σ^i is therefor described as follows:

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

The reward $\hat{r}$ is chosen so it maximizes the cross-entropy loss between the predictions and the human choices or labels. The loss is therefore defined as follows:

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

Some 'empirical' changes were made by the authors to this basic approach. These were discovered while performing some early experiments:

- The ensemble of predictors to fit is chosen from $\mathcal{D}$ and even though $|\mathcal{D}|$ triples are chosen, they are not exactly the set $\mathcal{D}$. The triples are chosen from it with replacement, meaning that the same triplet could be in the fitting ensemble more than once. Each of these predictors is independently normalized, and the estimate $\hat{r}$ is determined by averaging the results.

- A portion of the data is left out to be used as a validation set for each predictor. One of two regularization methods is also applied: dropout or more commonly $\mathcal{L}_2$ regularization. For the $\mathcal{L}_2$ regularization, the corresponding coefficient is adjusted so the validation loss is from 1.1 to 1.5 times the training loss.

- Since humans can constantly make mistakes, it is assumed that there is a 10% chance that humans respond uniformily at random and this is consequently adjusted.

Finally, we address the queries. The decision on how to query preferences is based on an approximate estimate of the uncertainty in the reward function. The authors take the latest agent-environment interaction and get a large number of pairs of trajectories of length k, then they use each reward predictor in the ensemble to predict which segment from each pair is preferred. This is a rough approximation, and some of the experiments showed that it actually degrades performance in some tasks. Ideally, this approximation should be replaced by the expected value of the information in the query, but the authors leave this to further exploration.

4. Authors' Experiments

In this section, we focus on the implementation of the algorithm. Two different experiments were performed in the original paper, we explain each and we show the results our own implementation of one the experiments. The authors implemented the algorithm with TensorFlow and used the environment Mujoco and Arcade Learning from OpenAI Gym.

4.1 Reinforcement Learning Tasks with Unobserved Rewards

Initially, the authors conducted experiments on a variety of classical deep reinforcement learning tasks with unobserved rewards. Instead of being informed about the true objective, the agent was presented with pairs of trajectory segments, and it learned about the purpose of the task by querying humans about which segment is preferable. The authors sought to solve this task quickly while using as few queries as feasible. Contractors were used to provide feedback, and they were given a brief task description before being asked to compare several hundred to a few thousand pairs of trajectory segments for the task. The experiments involving real human feedback required between 30 minutes and 5 hours of human time. Real human feedback is capable of outperforming traditional reinforcement learning in some instances because it has the potential to provide a more well-defined reward. The authors also conducted experiments using synthetic oracles with preferences over trajectories that accurately reflected the reward in the underlying task. The details of the experiments are described in more detail, including model architectures, environmental modifications, and reinforcement learning algorithms used to optimize the policy.

4.1.1 Simulated robotics

The authors evaluated their proposed deep reinforcement learning algorithm by conducting experiments on eight simulated robotics tasks in MuJoCo. Modifications were made to these tasks to prevent encoding information about the task in the environment. They included a simple cartpole task for comparison purposes. The reward functions for these tasks were linear functions of distances, positions, and velocities and all were quadratic functions of features. The results showed that with 700 human labels, the algorithm could match traditional reinforcement learning on all tasks. While training with learned reward functions had comparable mean performance, it was less stable and had higher variance. Interestingly, the algorithm performed slightly better with 1400 labels than if it had been given the true reward, possibly because the learned reward function was better shaped. The efficiency of human feedback varied depending on the task, but it was typically only slightly less effective than synthetic feedback. In the Ant task, human feedback outperformed synthetic feedback, possibly because the feedback encouraged the robot to remain upright, which was a useful reward shaping technique.

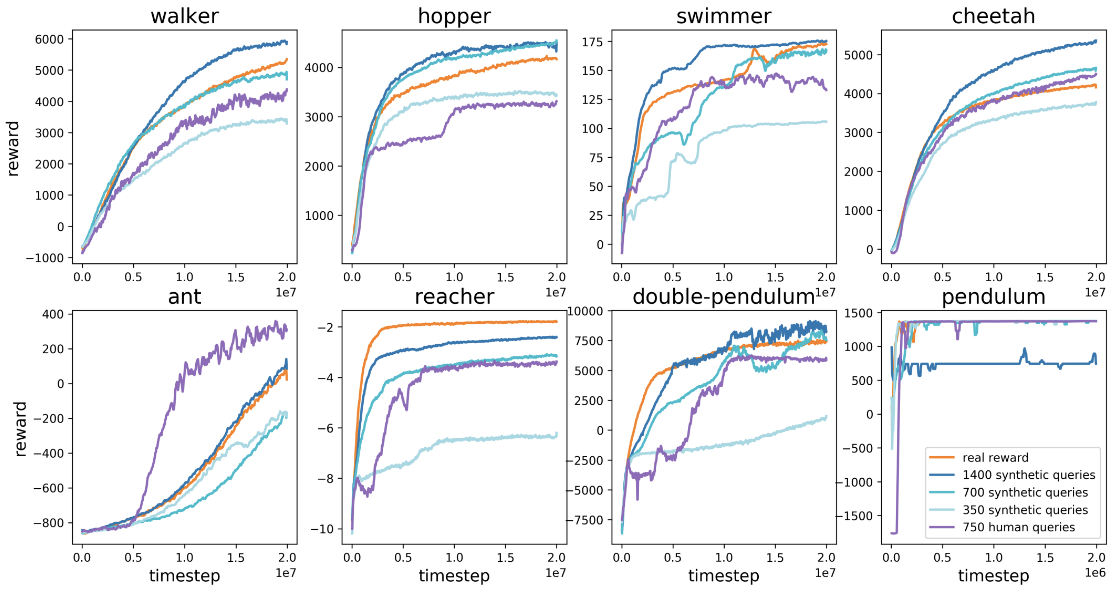

Figure 2: Results on MuJoCo simulated robotics

4.1.2 Atari

The researchers focused on a set of seven Atari games in the Arcade Learning Environment and compared the results of training their agent with different types of feedback: 5,500 queries to a human rater, synthetic queries (350, 700, or 1,400), and RL learning from the real reward.

While their method did not match RL for the challenging environments, it showed significant learning on most tasks and even matched or exceeded RL on some, such as BeamRider and Pong, where the results using synthetic labels were close to RL even with only 3,300 labels.

On Seaquest and Qbert, they learned slower using synthetic feedback, but still achieved an almost even performance level of RL. On SpaceInvaders and Breakout, synthetic feedback did not compete with RL, but the agent still improved noticeably, usually completing the first level in SpaceInvaders and reaching a score of 20 on Breakout, or 50 with enough labels.

Interestingly, real human feedback performed even or a little worse than synthetic feedback with 40 percent fewer labels. This may be due to errors made by the human overseers in labeling, leading to inconsistency between different contractors labeling the same run. In some cases, the uneven rate of labeling by contractors may also lead to labels overly concentrated in narrow parts of the state space.

The researchers suggest that future improvements to the pipeline for outsourcing labels may help address these issues. However, their method did not seem to work on Qbert, as they were not even able to beat the first level with real human feedback. They explain that this may be due to the use of short clips in Qbert, which can be confusing and difficult to evaluate, leading to confusing results for the agent to learn.

Finally, the researchers found that Enduro was difficult to learn for A3C due to the challenge of successfully getting other cars through random exploration. It seemed difficult to teach with synthetic labels, but over time, human labelers tended to reward any progress towards passing cars, shaping the reward and outperforming A3C in this game. This resulted in results comparable to what they would get with Deep Q Learning (DQN).

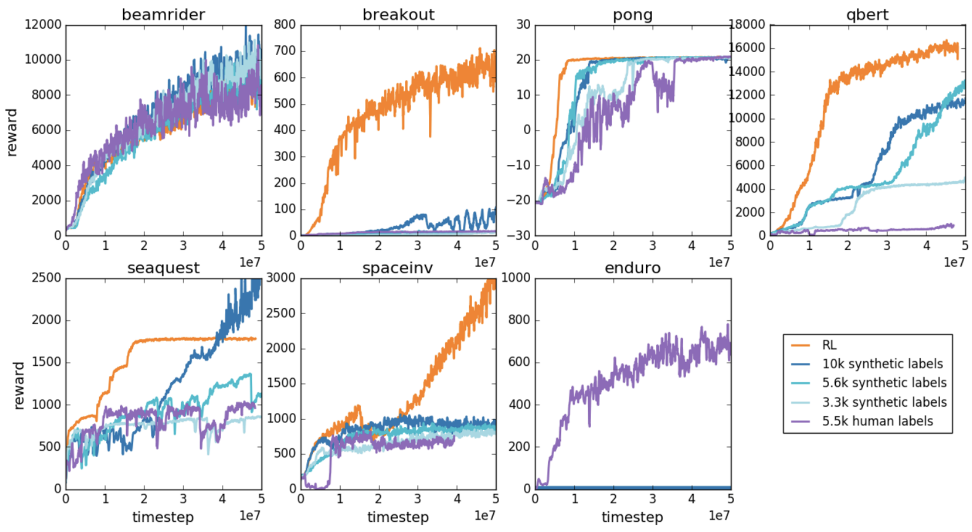

Figure 3: Results on Atari games

4.2 Novel behaviors

The purpose of human interaction in reinforcement learning is to solve tasks for which there is no available reward function. To demonstrate the effectiveness of their method in learning novel complex behaviors, the authors conducted experiments using the same parameters as the previous experiments. The following behaviors were learned:

1. The Hopper robot performing backflips consistently and repetitively with the use of 900 queries in less than an hour.

2. The Half-Cheetah robot moving forward while standing on one leg with the use of 800 queries in under an hour.

3. Keeping alongside other cars in Enduro for a substantial fraction of the episode using roughly 1,300 queries and 4 million frames of interaction with the environment, although it gets confused by changes in background.

4.3 Ablation Studies

To evaluate the algorithm's performance, we examine several modifications:

1. Instead of prioritizing queries with disagreement, we choose queries uniformly at random (random queries).
2. We use only one predictor instead of an ensemble, resulting in queries being randomly selected (no ensemble).
3. We only use queries collected at the beginning of the training process rather than throughout (no online queries).
4. We remove the '2 regularization and only use dropout (no regularization).
5. On the robotics tasks exclusively, we limit the length of trajectory segments to 1 (no segments).
6. We compare using the true total reward over a trajectory segment provided by an oracle, rather than fitting $\hat{r}$ through comparisons. We fit $\hat{r}$ using mean squared error (target).

Figure 4 presents the results for MuJoCo, while Figure 5 presents the results for Atari. Offline reward predictor training results in poor performance due to nonstationarity of occupancy distribution. This leads to partial reward maximization and bizarre behavior. Human feedback needs to be intertwined with RL learning rather than provided statically. Comparisons are easier to elicit than absolute scores, and for continuous control tasks, predicting comparisons works better than predicting scores. Longer clips are more helpful per clip, but it takes longer to evaluate them. Comparing longer clips is often easier in Atari environments since they provide more context than single frames.

5. Our Experiments

In this section, we focus on the implementation of our algorithm using the CartPole environment from OpenAI Gym and TensorFlow.

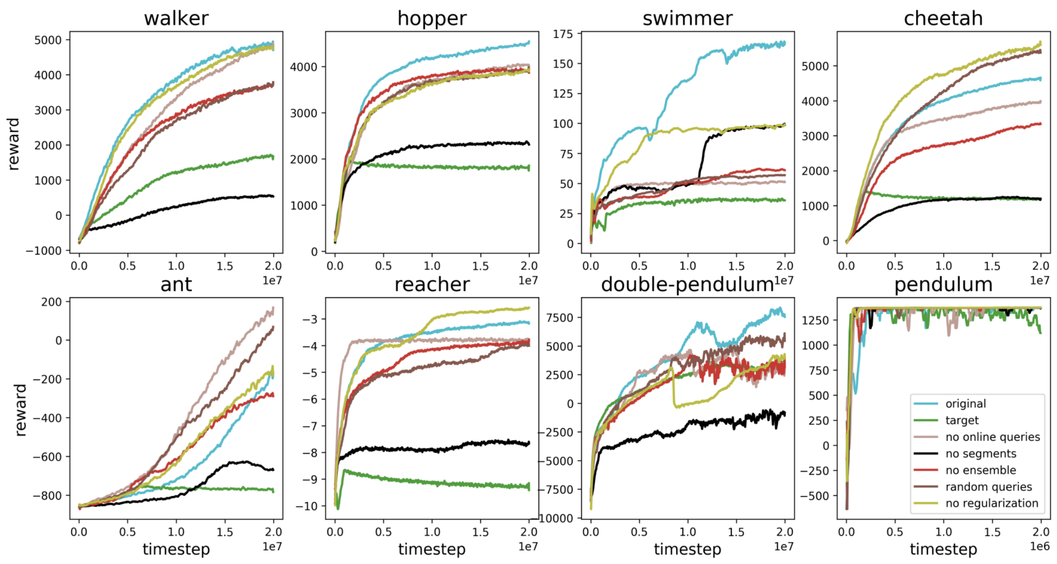

Figure 4: Performance of the algorithm on MuJoCo tasks after removing various components

5.1 CartPole Problem

We opted to experiment with these techniques using the CartPole problem, a prevalent challenge in reinforcement learning. The objective of this game is to maintain the balance of the pole for as long as feasible. To accomplish this, the system is managed by administering a force of either +1 or -1 to the cart. With every action performed, the environment provides multiple parameters for evaluation, including the cart's position and velocity, the pole's angle, and the pole's velocity.

We made three different experiments:

1. With Human Preferences
2. Without Human Preferences
3. With Human Preferences and fitting the Reward function.

The CartPole problem is defined by:

Action Space :

The action can take two values: 0, 1 indicating the direction of the fixed force the cart is pushed with. 0 corresponds to 'Push cart to the left' and 1 to 'Push cart to the right'.

The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it.

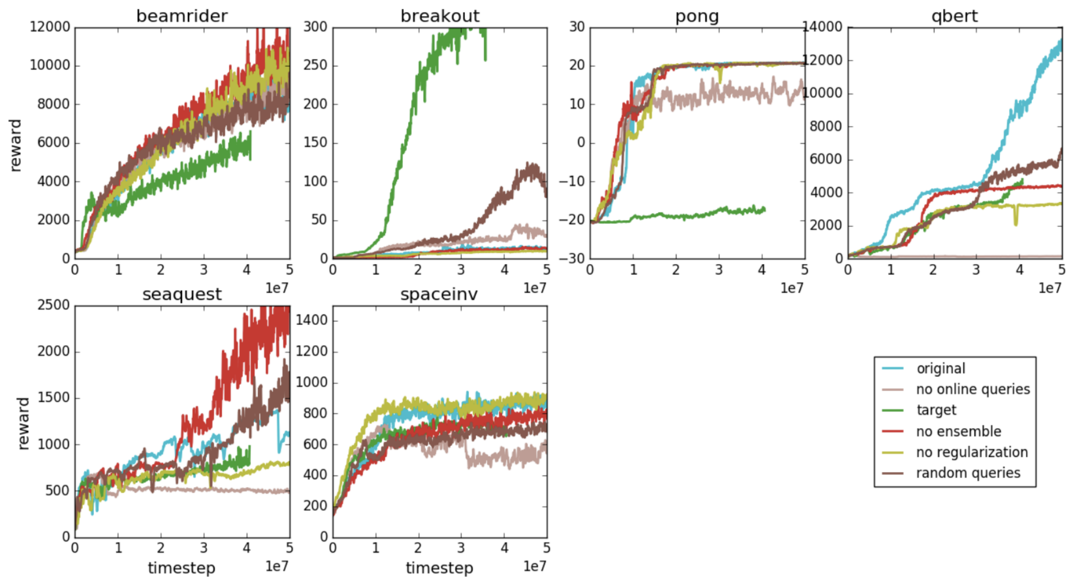

Figure 5: Performance of the algorithm on Atari tasks after removing various components

Observation Space :

The observation can take four values corresponding to the following positions and velocities:

0. Cart Position = [-4.8, 4.8]
1. Cart Velocity = [-Inf, Inf]
2. Pole Angle = [-0.418 rad (-24°), 0.418 rad (24°)]
3. Pole Angular Velocity = [-Inf, Inf]

Rewards :

Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted.

Starting State :

All observations are assigned a uniformly random value in (-0.05, 0.05)

Episode End :

The episode ends if any one of the following occurs:

- Termination: Pole Angle is greater than ±12°

- Termination: Cart Position is greater than ±2.4 (center of the cart reaches the edge of the display)

- Truncation: Episode length is greater than 500 (200 for v0).

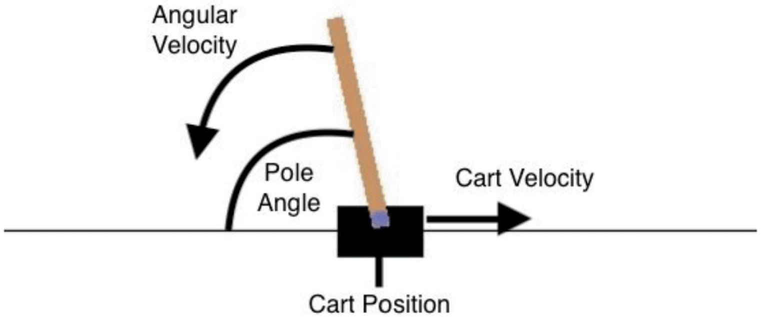

Figure 6: CartPole Problem

5.1.1 With Human Preferences

Our implementation trains an agent to play the CartPole game using an Actor-Critic Reinforcement Learning (RL) algorithm.

In the Actor-Critic RL algorithm, the agent is composed of two components: the Actor and the Critic. The Actor learns to take actions that maximize the expected cumulative reward, and the Critic learns to predict the expected cumulative reward of a given state.

The Actor class in the code defines the policy of the agent. The policy is a function that maps from the current state of the environment to a probability distribution over possible actions. The policy is represented by a neural network that takes in the current state and outputs a probability distribution over possible actions.

The stepepisode function performs a single episode of the game. In each time-step of the episode, the agent observes the current state of the environment, selects an action based on its policy, and receives a reward from the environment. The reward is then used to update the agent's policy.

The completeloss function calculates the loss based on the agent's actions. The loss is a measure of how far the agent's predicted action probabilities are from the action probabilities that would have been optimal in hindsight. The loss function used in the code is the Negative Log Likelihood (NLL) loss, which is given by:

$$\mathcal{L}(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) Q(s_t, a_t)$$

where θ represents the parameters of the policy, $\pi_\theta(a_t|s_t)$ represents the probability of taking action a_t in state s_t, and $Q(s_t, a_t)$ is the estimated value of taking action a_t in state s_t, which is provided by the Critic. The NLL loss is used because the aim of the agent is to maximize the expected cumulative reward, and the expected cumulative reward is proportional to the product of the probabilities of the chosen actions. Maximizing the product of probabilities is equivalent to minimizing the negative log of the product of probabilities, which is the NLL loss.

The calcg function calculates the discounted future reward (i.e. Q values) for each time-step in the episode. The discounted future reward is the sum of future rewards that the agent expects to receive, discounted by a factor of γ for each time-step into the future. The formula for the discounted future reward is given by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned}$$

where G_t is the discounted future reward at time-step t, R_{t+1} is the reward received at time-step $t+1$, and γ is the discount factor that determines the importance of future rewards relative to immediate rewards.

The Adam optimizer is used to update the parameters of the neural network. The Adam optimizer computes adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. This helps to accelerate convergence and improve the robustness of the training process.

Finally, the askhumanpreference function is a user-defined function that prompts a human to compare the current action taken by the agent and a random alternative action. It takes as input the current state, current action, and the next state obtained by taking the current action. It then selects a random alternative action, determines the state that results from taking this alternative action, and prompts the user to compare the two states based on their preference. The function returns the reward based on the human's preference, which is calculated as the difference in the user's preference between the current action and the random alternative action.

5.1.2 Without Human Preferences

Replacing the stepepisode function and the main code, we deleted the Human Preferences part. Our results demonstrate that using agent-environment interactions can be more cost-effective than using human interaction. Indeed, experiments involving real human feedback required between 30 minutes and 5 hours of human time. Regrettably, the authors did not furnish their database, which would have allowed for the comparison of various segments. However, the lack of information regarding the selection of sequences makes it challenging to replicate this database. As a result, We are unable to reproduce the findings of this article, but we nonetheless outlined the steps that should have been taken for implementing this approach.

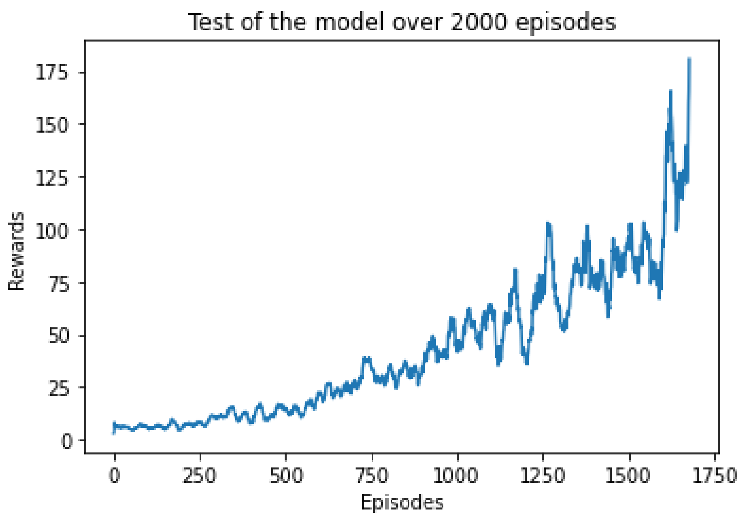

Figure 7: Final results, rolling average over the 20 past values

5.1.3 With Human Preferences and fitting the Reward function

The algorithm used is PPO (Proximal Policy Optimization) and consists of two main functions:

- stepepisode: Runs the model to get action logits and probabilities to select the next action. The selected action is used to calculate the reward and the state is updated accordingly. The function returns the action probabilities, rewards, states and actions of the episode.

- preferenceupdate: Based on two randomly selected transition points of the episode, the user inputs a preference (a:left, d:right, s:same) and the loss is calculated based on the rewards of the two transitions and the preference. This is used to update the reward function:

$$\hat{P}[\sigma_1 \succ \sigma_2] = \frac{\exp\left(\hat{r}(o_t^1, a_t^1)\right)}{\exp\left(\hat{r}(o_t^1, a_t^1)\right) + \exp\left(\hat{r}(o_t^2, a_t^2)\right)}$$

We write $\sigma_1 \succ \sigma_2$ to indicate that the human preferred trajectory segment σ_1 to σ_2.

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

$$loss(\hat{r}) = -\sum_{(\sigma_1,\sigma_2,\mu)\in D} \mu(1)\log \hat{P}[\sigma_1 \succ \sigma_2] + \mu(2)\log \hat{P}[\sigma_2 \succ \sigma_1]$$

Unfortunately, after a lot of research, we could not understand the problem related to the error obtained in the part of the code where we calculate the gradient to fit the reward function.

6. Discussion and Conclusion

In this report, we discussed the paper 'Deep Reinforcement Learning from Human Preferences'. The study showed that by using a separate reward model developed through supervised learning, the complexity of the interaction process can be reduced by approximately 3 orders of magnitude. This makes agent-environment interactions much more cost-effective than human interaction alone, and provides the first evidence that techniques for preference elicitation and reinforcement learning from unknown reward functions can be scaled up to state-of-the-art RL systems. The results of the study have important implications for the practical application of deep RL in complex real-world tasks, as it suggests that powerful RL systems can be trained using human preferences and used to address complex human values tasks. Future research can focus on improving the efficiency of learning from human preferences, expanding the range of tasks to which it can be applied, and making learning a task from human preferences as easy as learning it from a programmatic reward signal. Overall, the study provides a valuable framework for the use of programmatic reward signals to enable powerful RL systems to complete complex human value tasks, which has significant implications for the future of AI and human-computer interaction.

7. References

[1] Ayush Thakur. "Understanding Reinforcement Learning from Human Feedback (RLHF): Part 1".

[2] Daniel Eid. 'Deep Reinforcement Learning From Human Preferences in tensorflow'.

[3] Riad Akrour et al. "Programming by feedback".

[4] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents".

[5] Greg Brockman et al. "Openai gym".

[6] Dylan Hadfield-Menell et al. “Cooperative inverse reinforcement learning”.

[7] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”.

[8] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.”

[9] John Schulman et al. “Trust region policy optimization”.

[10] Akrour, R., Schoenauer, M., and Sebag, M. (2011). Preference-Based Policy Learning. ECML/PKDD.

[11] Pilarski, Patrick M., et al. ”Online Human Training of a Myoelectric Prosthesis Controller Via Actor-critic Reinforcement Learning.” (2011).

[12] P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey and R. S. Sutton, ”Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning,” (2011)

[13] J. Secretan et al., ”Picbreeder: evolving pictures collaboratively online.” (2008).

[14] A. T. Sørensen, et al. ”A robust activity marking system for exploring active neuronal ensembles” (2016).

[15] Akrour, R., Sorokin, D., and Peters, J. ”Policy learning with continuous state and action representations”. (2012).

[16] Wilson, A. G., Chu, W., and Pietquin, O. ”Using human feedback for interactive reinforcement learning”. (2014).