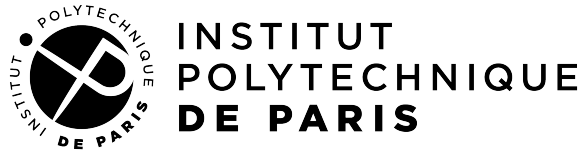
Study of the article: Deep Reinforcement Learning from Human Preferences

Audrey AIRAUD and Nathan TOUMI



M2DS Institut Polytechnique de Paris

February 28, 2025

Contents

1 Introduction **1**

2 General Framework and Terminology **2**

3 Previous Related Work **4**

4 Description of the Method **7**
4.1 Formal Problem Definition 7
4.2 Framework Overview 8
4.3 Algorithms to optimize the policy (Step 1) 9
4.3.1 Advantage actor-critic (A2C) 9
4.3.2 Trust Region Policy Optimization (TPRO) 11
4.4 Human preferences (Step 2) 12
4.5 Fitting the reward function (Step 3) 12

5 Experimental Analysis **14**
5.1 Analysis of Published Experiments 14
5.1.1 Simulated Robotics 15
5.1.2 Atari Games 15
5.1.3 Ablation studies 16
5.1.4 Novel behaviors 17
5.2 Our Experiments 17
5.2.1 Implementation of Advantage Actor-Critic 18
5.2.2 Implementation of Human preferences 20

6 Discussion and conclusions **24**

Chapter 1

Introduction

The aim of this work is to analyze the article *Deep Reinforcement Learning from Human Preferences* [1], highlighting its main contributions. Published in 2017, the article starts from the observation that solving a reinforcement learning (RL) task in a real environment is often highly complex, as defining and demonstrating the desired behavior can be challenging. To overcome this limitation, the authors focus on solving RL tasks without direct access to the reward function. Instead, they propose allowing a human to supervise the task by providing feedback on the system's behavior.

However, a major challenge arises: using human feedback directly as a reward function is extremely time-consuming. Therefore, the authors also aim to significantly reduce the amount of human feedback required while maintaining high learning efficiency.

In this study, we will first review prior research related to this topic. Next, we will detail the methodology proposed in the article. We will then explore subsequent work that has built upon this research. Following this, we will examine the experimental results, first by analyzing the experiments presented in the article and then by discussing our own experimental findings. Finally, we will summarize our study and present our conclusions.

Concerning our experiments, the codes can be found in the following Github repository:
`https://github.com/odd2022/Deep-Reinforcement-Learning-From-Human-Preferences-M2DS`

Chapter 2

General Framework and Terminology

Before delving into the subject, we wanted to recall and define the concepts specific to our study. Once defined, we will use them throughout our study. At the end of these definitions, we will briefly describe the general idea of the article we are studying.

The article we are studying presents a reasoning approach that falls within the framework of Reinforcement Learning. Reinforcement learning is a type of machine learning where an agent interacts with an environment to solve tasks. Unlike supervised learning, which relies on labeled datasets, RL operates sequentially: at each time step, the agent selects an action, interacts with the environment, and receives feedback in the form of a reward. This reward signal helps the agent evaluate the quality of its actions and adjust its decision-making policy accordingly.

In RL, the agent's actions influence future states. The learning process involves trial and error—initially, the agent starts without prior knowledge and gradually improves by exploring the environment and optimizing its behavior based on the received rewards. This learning framework differs from traditional machine learning models, which typically predict outputs from a fixed dataset. Instead, RL algorithms aim to maximize cumulative rewards over time, improving their strategies through interaction with the environment. The agent learns to make optimal decisions through reinforcement signals, either positive or negative, leading to continuous improvement.

Key Concepts in Reinforcement Learning

- Environment: The external system with which the agent interacts. It defines the possible states the agent can be in and determines the consequences of the agent's actions.
- Agent: The entity that learns how to behave within the environment by selecting actions.
- Action: A decision made by the agent at a given time step k, influencing the next state of the environment.
- State: A representation of the current situation of the environment.
- Observation: The information available to the agent about the environment at each time step.

- Reward: A numerical signal that provides feedback on the quality of an action taken by the agent. The reward serves to distinguish between favorable and unfavorable outcomes for the agent, guiding the learning process by offering feedback on the effectiveness of various actions. The goal of the agent is to maximize the cumulative rewards over time.
- Policy: The strategy used by the agent to select actions based on the current state or observation. The policy evolves as the agent learns from interactions with the environment. An optimal policy should not only consider immediate rewards but also long-term benefits. This is captured by the value function, which estimates the expected cumulative reward from a given state, helping the agent prioritize actions that lead to better long-term outcomes.

Deep Reinforcement Learning This paper explores a Deep RL architecture, which integrates artificial neural networks into the RL process. The use of Deep Learning in Reinforcement Learning provides several key advantages, primarily by enhancing the ability to map states and actions to rewards more effectively. Neural networks enable function approximation, allowing agents to generalize across large or continuous state spaces where traditional tabular approaches would be infeasible. Additionally, DRL facilitates policy optimization, improving the efficiency of learning strategies and decision-making in complex environments. By leveraging deep learning techniques, RL models can scale to more intricate tasks, achieving better generalization and adaptability.

General Idea of Deep Learning from Human preferences Now that we have defined the key concepts of Reinforcement Learning, we can introduce the problem.

As previously discussed, Reinforcement Learning typically depends on a predefined reward function to guide the agent's learning process. However, in many real-world applications, designing a reward function that is both explicit and accurate is very complex.

To address this issue, the algorithm presented in the article learns the reward function dynamically rather than relying on a manually defined one. Instead of assigning fixed rewards, the method collects feedback from a human user. The system presents short video clips showcasing different agent behaviors, and the human evaluator selects the one that best aligns with their preferences. This feedback is then used to refine a reward predictor, which progressively approximates the true reward function.

As the reward function evolves, the agent's policy is continuously trained to optimize its behavior based on this learned reward signal. This process allows the agent to align its actions more closely with human expectations, overcoming the limitations of predefined reward functions.

Chapter 3

Previous Related Work

In the literature, many studies have explored human involvement in artificial learning processes. Below are several approaches that integrate human feedback without necessarily falling under reinforcement learning.

Global Learning Methods with Human Feedback

- *Picbreeder: Evolving pictures collaboratively online* [2] describes Picbreeder, an image-generation algorithm that evolves based on user preferences. Unlike reinforcement learning, there is no autonomous agent exploring an environment; instead, images evolve purely through genetic algorithms, guided solely by human selections. The algorithm used is NEAT (NeuroEvolution of Augmenting Topologies), which starts with a simple neural network and gradually increases in complexity by adding neurons and connections through genetic mutations. This article shares the concept of learning from user preferences with the one we study: in both cases, the system improves iteratively as users select their preferred generated outputs.

- *A Bayesian Interactive Optimization Approach to Procedural Animation Design* [3] This article focuses on computer graphics and animation generation, specifically on optimizing animation settings that control how characters and objects move, such as realistic physics-based motion or walking animations.. The goal is to refine animations according to user preferences. To achieve this, the system employs a probabilistic model (Gaussian Process Regression) that estimates how different animation parameters impact animation quality. The system then asks a human evaluator to compare different animations, and this feedback is used to iteratively improve the model. This is an example of a learning process where humans are actively in the loop, guiding the optimization rather than letting the system learn autonomously. Unlike reinforcement learning, it does not involve an agent optimizing a policy via rewards but instead uses Bayesian inference to fine-tune static animation parameters efficiently.

In the field of Reinforcement Learning, several prior studies have explored the role of human feedback. According to the article we are studying, two main approaches stand out: some methods use human ratings or rankings to guide learning, while others rely on preference-based approaches, where the system learns from comparisons between two generated behaviors rather than absolute ratings.

Reinforcement Learning from Direct Human Ratings/Rankings:

- *Preference-Based Policy Learning (PPL)* and *APRIL: Active Preference-learning based Reinforcement Learning* [4] are approaches designed for reinforcement learning when an explicit reward function is unavailable. In these cases, human experts may still provide preferences by ranking different agent behaviors. The learning process follows an iterative cycle consisting of four steps. First, the agent demonstrates a policy, which is then ranked by the expert in comparison to previous demonstrations. Based on these rankings, the system learns a policy return function, allowing it to evaluate and improve its policies. Using this updated information, the agent generates a new candidate policy, and repeats the process iteratively until it has the right behavior. This mechanism provides a valuable learning process without requiring a numerical reward function.

  APRIL builds on *PPL* but introduces active learning techniques to reduce the number of times the expert needs to rank policies. Instead of randomly selecting policies for ranking, the system intelligently chooses the most useful ones for the expert to evaluate. This is done using a method called Approximate Expected Utility of Selection, which helps the system identify the policies that will provide the most valuable information for learning. By focusing only on the most relevant rankings, APRIL makes the learning process more efficient and reduces the expert's work.

- *A Bayesian Approach for Policy Learning from Trajectory Preference Queries* [5] uses a Bayesian framework and trajectory preference queries for policy learning without relying on a numerical reward function. The expert compares trajectory pairs to guide the agent, with the goal of minimizing expert queries. The agent does not have direct access to the expert's ideal policy (latent policy) but instead estimates it probabilistically using Bayesian inference, updating its posterior distribution with each new expert comparison.

  Although the system appears to rely on preferences, as the expert selects the better trajectory, it actually constructs a global ranking of policies by aggregating multiple comparisons. The Bayesian model infers an underlying policy ranking, making this approach closer to ranking-based RL than pure preference-based optimization.

  The authors of the article we study explain that their method follows closely the method chosen in Wilson et al. [5], with some differences. Indeed, Wilson et al. assume that there is a "target policy" and the algorithm learns a reward function that captures how close a given policy is to this target, and they try to reconstruct an ideal behavior by comparing different trajectories to it. In comparison, the article we study learns only from human feedback, allowing it to develop new behaviors that might even surpass human intuition. Moreover, instead of performing full reinforcement learning, Wilson et al. fit the reward function and generate trajectories based on the MAP estimate of the target policy. Finally, their experiments rely on synthetic feedback generated from their model, while the approach of the article we study collects real human feedback from non-expert users. Additionally, the article we study is designed to perform more complex tasks, whereas it remains unclear if Wilson et al.'s method can generalize beyond simpler domains.

Reinforcement Learning from Human Preferences

- *Programming by Feedback* [6] is an interactive optimization framework where a system learns by iteratively generating solutions and receiving binary feedback from a human user. Instead of relying on a predefined reward function, the system presents a new candidate solution, which the user compares to the best-known solution so far and provides a preference judgment. Using this feedback, the system updates its utility function to better approximate the user's preferences and generates improved solutions in subsequent iterations. Unlike ranking-based approaches that construct a global ordering of policies, PF focuses on local comparisons to refine the search for an optimal solution.

  The authors of the article we study explain that their approach is close to this approach but extends it to more complex, high-dimensional tasks using modern deep RL techniques. While Akrour et al. focus on small discrete and low-dimensional continuous domains, assuming that the reward function is linear in hand-coded features, Deep Learning from Human Preferences removes these constraints by learning a fully nonlinear reward model using deep neural networks. Another key difference lies in how human preferences are collected—Akrour et al. use whole-trajectory comparisons, which are cognitively demanding, whereas Deep Learning from Human Preferences collects a much larger number of short-clip comparisons, making the process more efficient while requiring less total human time. Additionally, Deep RL from Human Preferences incorporates asynchronous training, ensembling, and scalable deep RL methods, allowing it to handle complex environments like physics-based robotic control and Atari games, which would be infeasible under the assumptions of Akrour et al..

- *Preference-based reinforcement learning: A formal framework and a policy iteration algorithm.* [7] develops a RL framework that does not require explicit numerical rewards, but instead learns policies from qualitative preferences between trajectories, actions, or policies. Unlike standard reinforcement learning, where policies are improved by maximizing expected cumulative rewards, their system learns relative action preferences within each state. Instead of assigning explicit value estimates to actions, the system generates multiple candidate actions, observes their consequences through rollouts, and ranks them based on qualitative comparisons provided by an expert or a preference model. This ranking is then used to iteratively improve the policy.

  The article we study uses pairwise comparisons of short trajectory clips to train a deep reward model, which is then used in a standard RL loop. In contrast, PBRL directly operates on preferences without learning a reward function, instead using policy iteration with ranked actions to optimize decision-making. Additionally, Deep Learning from Human Preferences is designed for large-scale deep RL problems like Atari or robotics, while PBRL is more focused on structured decision-making where ranking actions can be explicitly modeled.

To conclude this part, the authors of the article we study emphasize that their work leverages deep learning techniques to generalize across high-dimensional environments, whereas previous methods were largely constrained to low-dimensional tasks. This follows the broader trend of scaling reward learning methods in deep learning, as observed in areas such as inverse reinforcement learning and imitation learning. By incorporating human preference feedback into Deep Reinforcement Learning architectures, their approach enhances scalability and flexibility, making it applicable to more complex learning scenarios.

Chapter 4

Description of the Method

We have outlined the purpose and overall functioning of our system in the preliminary section. Now, in this chapter, we will formally define it and provide a detailed explanation of each component of the process.

4.1 Formal Problem Definition

We consider an agent interacting with an environment. At each time step t, the agent receives an observation $o_t \in \mathcal{O}$ from the environment and selects an action $a_t \in \mathcal{A}$, according to a policy $\pi : \mathcal{O} \to \mathcal{A}$. The selected action is then executed in the environment. This interaction results in a new state and observation, forming a sequential decision-making process.

As said in the preliminary section our approach relies on human feedback to infer the reward function. Instead of directly optimizing a predefined objective, the agent collects **trajectory segments**, which are sequences of observation-action pairs extracted from complete trajectories.

A trajectory τ represents a full sequence of interactions between the agent and the environment over a given episode:

$$\tau = ((o_0, a_0), (o_1, a_1), \ldots, (o_T, a_T)) \in (\mathcal{O} \times \mathcal{A})^{T+1}. \tag{4.1}$$

However, rather than evaluating entire trajectories, we focus on *trajectory segments*, which are sub-sequences of length k extracted from τ:

$$\sigma = ((o_t, a_t), (o_{t+1}, a_{t+1}), \ldots, (o_{t+k-1}, a_{t+k-1})) \subset \tau. \tag{4.2}$$

These segments provide a more manageable way to assess agent behavior, as they capture short-term interactions that can be compared more easily by a human supervisor.

A human supervisor is then asked to express preferences over pairs of trajectory segments. Given two trajectory segments σ_1 and σ_2, we use the notation:

$$\sigma_1 \succ \sigma_2 \tag{4.3}$$

to indicate that the human prefers trajectory segment σ_1 over σ_2. This **qualitative** preference can be interpreted **quantitatively** using an unknown reward function $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$. Specifically, if the human preferences are assumed to follow an underlying reward model, we have:

$$((o_0^1, a_0^1), \ldots, (o_{k-1}^1, a_{k-1}^1)) \succ ((o_0^2, a_0^2), \ldots, (o_{k-1}^2, a_{k-1}^2)) \tag{4.4}$$

whenever the cumulative reward along σ_1 exceeds that of σ_2:

$$\sum_{t=0}^{k-1} r(o_t^1, a_t^1) > \sum_{t=0}^{k-1} r(o_t^2, a_t^2). \tag{4.5}$$

However, as said before, we do not always have access to the explicit reward function in some cases. A key advantage of this approach is that it enables agent training in the absence of an explicit numerical reward function. In this case, the learning process is purely **qualitative**, as humans can express preferences based on subjective or high-level criteria that are difficult to formalize quantitatively. In the absence of an explicit reward function, qualitative evaluation is conducted by setting a goal expressed in natural language and having a human assessor judge how well the agent's behavior aligns with that goal based on observed trajectories or video demonstrations.

The core objective of this framework is to train an agent that generates trajectories preferred by humans while minimizing the number of queries required. Since human feedback is a limited and costly resource, the method must efficiently use the collected preferences to refine the policy.

4.2 Framework Overview

The proposed method consists of three distinct processes. At each time step, a policy $\pi : \mathcal{O} \to \mathcal{A}$ governs the agent's actions, while a reward function $\hat{r} : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ provides an estimate of the reward. These processes can be described as follows:

1. The policy π interacts with the environment to generate a set of trajectories. The parameters of this policy are then updated using a standard RL algorithm to maximize the cumulative predicted rewards, denoted as $r_t = \hat{r}(o_t, a_t)$.

2. Based on the trajectories generated in step (1), the algorithm selects a pair of trajectory segments (σ_1, σ_2) and presents them to a human evaluator for comparison.

3. Using the comparisons, a supervised learning algorithm is employed to update the parameters of the reward function $\hat{r}$, to better align with the collected human preferences.

These processes operate asynchronously, forming a continuous learning loop. The method begins with step (1), where trajectories are generated, then moves to step (2), where human evaluations are collected. Subsequently, in step (3), the reward function $\hat{r}$ is updated based on the human comparisons. The refined reward function is then fed back into step (1), and the cycle repeats.

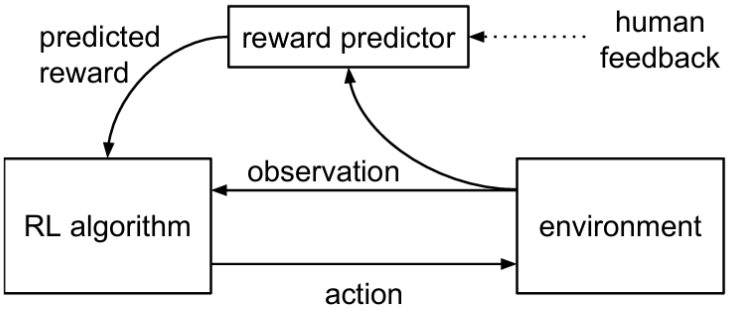


Figure 4.1: Overall architecture proposed

4.3 Algorithms to optimize the policy (Step 1)

Once the reward function $\hat{r}$ has been used to assign rewards, the problem can be approached as a standard RL task. However, one challenge arises from the potential non-stationarity of $\hat{r}$, as human preferences evolve dynamically when selecting between trajectory segments. This characteristic makes it preferable to use RL algorithms that are robust to variations in the reward function.

To mitigate instability and enhance the performance of the RL algorithm, a preprocessing step is applied where $\hat{r}$ is standardized to have zero mean and constant standard deviation. This ensures that the reward remain consistent throughout training, preventing large fluctuations that could impact learning stability. Once standardized, an appropriate RL algorithm is selected depending on the task.

Given the need for adaptability to changing rewards, policy gradient methods have proven to be particularly effective in this context. These methods directly optimize the policy by adjusting its parameters through gradient-based updates, aiming to maximize the expected cumulative reward.

In their experiments, the authors employed different RL algorithms tailored to specific applications. For tasks involving Atari games, they used the Advantage Actor-Critic algorithm [8], using its ability to efficiently update policies in large state spaces. Meanwhile, for simulated robotics tasks, they adopted Trust Region Policy Optimization (TRPO) [9], a method designed to maintain stable policy updates within a defined trust region, ensuring reliable exploration. In both cases, the hyperparameters were set to values that have been effective in standard RL settings, with the entropy bonus for TRPO being the only parameter specifically adjusted to encourage adequate exploration in response to changes in the reward function.
In the following parts, we will describe these two algorithms.

4.3.1 Advantage actor-critic (A2C)

The A2C algorithm seeks to combine the strengths of both value-based and policy-based reinforcement learning methods. To better understand this approach, let's briefly review the two concepts.

Value-based RL methods focus on learning a value function, which estimates the expected cumulative reward (also known as the return) from a state or state-action pair. Algorithms such as Q-learning [10] or SARSA [11] fall into this category. In these methods, the policy is indirectly derived from the actions that maximize the expected return. Value-based methods are easy to implement, intuitive, and work well in environments with a finite action space. However, they struggle with high-dimensional action spaces, and since the policy is not explicitly represented, it's difficult to incorporate prior knowledge into the algorithm.

Policy-based methods, on the other hand, directly learn the policy without estimating a value function. The agent optimizes the policy parameters to maximize the expected return. The policy $\pi(a|s;\theta)$ is parameterized (often using a neural network) to map states to actions, and the optimization adjusts θ to maximize cumulative rewards. An example of a policy-based algorithm is TRPO, which will be explained later. These methods directly optimize the policy, which allows them to handle continuous action spaces, support stochastic policies, and explicitly represent the policy. However, they can be sample-inefficient, have high variance, and can be more complex to implement.

As mentioned earlier, A2C methods leverage the strengths of both value-based and policy-based approaches. To achieve this, the method involves an *actor*, which represents the policy responsible for selecting actions, and a *critic*, which evaluates the value function based on the feedback from the environment about the state.

Additionally, an *advantage function* is used to compute the advantage (also known as TD error or prediction error). The advantage function determines whether the actual reward is better or worse than expected. If an action leads to a negative advantage, its probability should decrease, while a positive advantage increases the likelihood of that action. This approach relies on temporal difference (TD) learning, meaning the agent learns by predicting future rewards and adjusting its actions based on the prediction error. This prediction error is calculated using the advantage function as follows:

$$TD_{error} = TD_{target} - V(S) \tag{4.6}$$

With:

- The value of a state $V(S)$ which corresponds to the output of the critic network. It represents the expected cumulative reward an agent can obtain from that state while following a given policy, helping the critic assess how favorable a state is for future rewards.
- The TD target, which is an estimate of the expected return and it is computed at each step using the current reward R, the discounted future rewards (weighted by γ), and the value of the next state $V(S')$. This results in the following formula for all steps except the final one:

  $$TD_{target} = R + \gamma V(S') \tag{4.7}$$

  For the final step, $TD_{target} = R$ since there is no subsequent state to account for in the future reward.

The actor network is a neural network that maps states to action probabilities. Given a state, it estimates the probability of selecting each possible action. The network's parameters are updated based on feedback from the agent. The objective is to adjust the weights of the neural network such that actions that do not contribute to achieving the goal become less likely over time, while actions that are beneficial are reinforced.

The critic network is a neural network that maps states to Q-values, which represent the value of a given state. Its output corresponds to the $V(S)$, so the final layer of the network consists of a single neuron. The goal is to minimize the TD_{error}, and therefore, the weights of the critic network are updated by optimizing the TD_{error} using Mean Squared Error as loss function.

4.3.2 Trust Region Policy Optimization (TPRO)

As previously discussed, this method is an example of a policy gradient approach, meaning it aims to optimize the cumulative reward by adjusting the policy to maximize the expected return. It uses gradient ascent to optimize the parameters, as the goal is to maximize the cumulative reward. The general concept of policy gradient methods involves collecting multiple trajectories from the current policy π_θ, computing the advantage A and the policy gradient g, and updating the policy parameters θ using the following ascent formula:

$$\theta = \theta + \alpha g$$

However, this method can lead to suboptimal actions, which may result in poor final states. Additionally, a single "bad" step can ruin the entire training process since the training is based on the current policy (non-stationary training).
To address this issue, TRPO was proposed as a solution. The idea behind TRPO is to take the largest possible step that improves the policy's performance. To achieve this, it computes the distance between the old and new policies by adding a constraint to the optimization process. The authors measure this difference using the Kullback-Leibler divergence [12], which is used to quantify how close two distributions are. The Kullback-Leibler divergence is computed as follows:

$$D_{\mathrm{KL}}(\pi_{\theta(\text{current})} \parallel \pi_{\theta+\Delta\theta(\text{new})}) = \sum_x \pi_{\theta(\text{current})}(x) \log\left(\frac{\pi_{\theta(\text{current})}(x)}{\pi_{\theta+\Delta\theta(\text{new})}(x)}\right) \tag{4.8}$$

The goal is to maximize a cost function while maintaining a constraint on the distance between the old and new policies. This leads to the following optimization problem:

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

This approach allows the gradient to take large steps in flatter regions, accelerating convergence, while making smaller steps in more curved regions. To ensure that the updates respect the KL-divergence constraint, TRPO uses a line search strategy, reducing the step size as needed. Unlike Natural Policy Gradients, which assume the constraint is satisfied, TRPO actively enforces it. Furthermore, TRPO verifies policy improvement by calculating advantages from the old policy and adjusting probabilities using importance sampling before accepting the update.

4.4 Human preferences (Step 2)

Once the policy is optimized (step 1 of our overall framework), we will integrate human feedback into the system. In this phase, the human observer will be presented with a short clip (lasting between 1 and 2 seconds) for each segment. Since the objective is to select a preference between two trajectories, the human will view two clips and specify which segment is better than the other, state that both segments are equally good, or assert that the two segments are incomparable.
All the data is stored in a database D as triples of the form $(\sigma_1, \sigma_2, \mu)$, where σ_1 and σ_2 are the two segments being compared, and μ represents the distribution over $\{1, 2\}$. The human can choose to assign all the mass to 1 (if segment σ_1 is preferred), all the mass to 2 (if segment σ_2 is preferred), or distribute the mass uniformly if both segments are considered equally good. If the two segments are deemed incomparable, the comparison is not recorded in the database.

We must remember that the goal is to minimize the number of questions asked in order to save human time, while still maintaining an acceptable level of performance. To do so, we have to know how to select queries, specifically how to determine which choices to present to humans. To achieve this, the auhtors of the article first extracted a large set of segment pairs of a predefined length. Then, for each pair, they used several predictors to predict which segment would be preferred. Afterward, they calculated the variance across all pairs. A large variance indicates significant uncertainty in the predictions, meaning the predictors provided different results. Consequently, the pairs with the highest variance are selected for human questioning, as the authors argue that high uncertainty could signal interesting cases worth further exploration. The intuition is that when there is doubt (a large variance) among the predictors, it would be beneficial to ask for human expertise to resolve the uncertainty. However, this approach is a rough approximation, and some experiments (which will be discussed in the next chapter) suggest that it can sometimes lead to worse outcomes.

4.5 Fitting the reward function (Step 3)

The authors interpret the reward function estimate $\hat{r}$ as a preference predictor, viewing it as a latent vector representing human judgments. They assume that the probability of preferring a segment σ_i is exponentially related to the summed latent reward over the entire clip shown to the human. The probability that σ_1 is preferred over σ_2 is modeled according to the Luce-Shepard choice rule [13]. This rule states that the likelihood of selecting one option over another is proportional to an exponential function of its value. In other words, the greater the relative value of an option compared to the alternatives, the higher the probability of it being chosen. In the context of human preference learning, the model assumes that the likelihood of preferring one segment over another depends on the total estimated reward associated with each segment. Instead of making absolute decisions, the model assigns probabilities to preferences, reflecting the uncertainty and variability in human judgments. The formula is given by:

$$\hat{P}[\sigma^1 \succ \sigma^2] = \frac{\exp \sum \hat{r}(o_t^1, a_t^1)}{\exp \sum \hat{r}(o_t^1, a_t^1) + \exp \sum \hat{r}(o_t^2, a_t^2)} \tag{4.9}$$

Now that we have a mathematical model for human preferences between two segments, the goal is to adjust the reward function $\hat{r}$ so that its predictions align as closely as possible

with human choices. To achieve this, we minimize a cross-entropy loss, defined as:

$$\text{loss}(\hat{r}) = -\sum_{(\sigma^1,\sigma^2,\mu)\in D} \mu(1)\log\hat{P}[\sigma^1 \succ \sigma^2] + \mu(2)\log\hat{P}[\sigma^2 \succ \sigma^1] \tag{4.10}$$

Fitting Process The reward function is fitted using a specific procedure. The authors adopted an approach similar to bagging, where multiple predictors are trained on different subsets $|D|$ drawn from the dataset D introduced in section 4.4. The final reward estimator $\hat{r}$ is obtained by independently normalizing each predictor and averaging their outputs. However, unlike traditional bagging, the extracted samples from D are not necessarily independent.

To enhance model performance, the authors applied regularization techniques, specifically dropout and L_2 regularization. For L_2 regularization, they introduced a validation step to optimize the regularization coefficient, ensuring a weighted L_2 normalization. A fraction $\frac{1}{e}$ of the training data is set aside as a validation set. The objective of this normalization is to maintain the validation loss within 1.1 to 1.5 times the training loss.

Finally, recognizing that human responses may occasionally contain errors, the authors assume that 10% of human feedback is random, following a uniform distribution. This inherent uncertainty is accounted for and incorporated into the model's adjustments.

Chapter 5

Experimental Analysis

In this chapter, we discuss the experiments related to the studied article. The first section provides an overview of the experiments conducted by the authors, while the second section details our own experimental approach.

5.1 Analysis of Published Experiments

As previously mentioned, the authors explored two distinct types of tasks: simulated robotics using the MuJoCo interface [14] and Atari games via the Arcade Learning Environment [15] within the OpenAI Gym framework [16]. This section aims to outline their studies and summarize their findings.

The advantage of the method presented so far is its ability to solve tasks without directly observing the reward. To demonstrate this, the authors trained their models using only human feedback. Their goal was to choose between two segments, each lasting between 1 and 2 seconds (as described in section 4.4). Depending on the experiment, the human time required to train the model ranged from 30 minutes to 5 hours.

The authors also conducted experiments based on synthetic preferences, based on the task reward (which, of course, is only possible for tasks where the reward is available). To maintain the same structure, rather than sending the comparison to a human, a synthetic oracle was used to select one of the two presented trajectories, choosing the one that maximized the task reward. Finally, they compared the actual reward with the results obtained through their method (using either human feedback or a synthetic oracle).

The original goal was not to outperform traditional methods but to approximate the reward as closely as possible without directly observing it. Interestingly, human feedback sometimes led to a better-shaped reward, resulting in more successful tasks than classic reinforcement learning with access to the reward.

In the next two sections, we describe the two environments used by the authors for their experiments and the results they obtained.

5.1.1 Simulated Robotics

The authors first considered eight simulated robotic tasks using the MuJoCo interface [14] within the OpenAI Gym framework [16]. The reward functions for these tasks are based on distances, velocities, and positions, with the goal of having the robot perform specific actions. To evaluate the effectiveness of their methods, the authors examined how the reward function evolved over time with various parameterizations. By comparing with the actual reward, the graph 5.1 shows the rewards for 700 human queries, as well as 350, 700, and 1,400 synthetic queries. The human queries were provided by contractors who were unfamiliar with the environments and the algorithm, except for Reacher and Cheetah, which were trained by the authors.

Generally, training with 700 queries captures the mean performance, although synthetic queries produce slightly less stable reward functions. However, with 1,400 synthetic queries, the learned reward function appears better-shaped, and the algorithm performs slightly better than when trained with the true reward function. However, when it comes to human feedback, it is generally less effective than synthetic feedback, often being only half as effective or, in the best cases, as good as the synthetic method. There is one exception: the Ant task, where human feedback proved more useful. In this case, humans favored when the robot was "standing upright," which turned out to be very beneficial for the task but was not well integrated into the reward function used for synthetic queries. Thus, we have seen that for these tasks, the value of human preferences is limited when the true reward is known. The ultimate goal, however, is to train an algorithm without access to the real reward. Despite this limitation, the performance remains relatively strong.

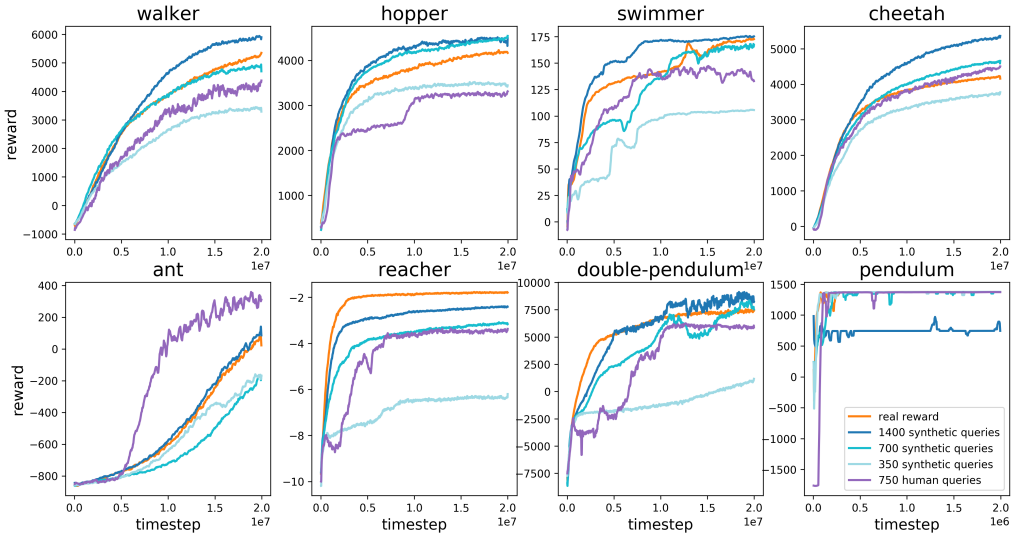


Figure 5.1: Results for simulated robotics tasks from the original paper

5.1.2 Atari Games

The authors then turned their attention to a well-known RL problem: the Atari Games. Introduced as a reference problem in the field by Bellemare et al. [15], it enables RL algorithms to be tested in challenging environments.

In figure 5.2, the evolution of the reward is shown as a function of time steps. The comparison was made between performance using 5,500 queries from a real human, 350, 700, or 1,400 synthetic queries, and finally, RL from the real reward. Several games were tested, including Space Invaders, Qbert, and others.

In most games, human feedback performed as well as, or worse than, synthetic feedback. Despite synthetic feedback having 40% fewer labels, it achieved comparable performance to human feedback. The authors propose that this may be due to human errors in labeling and possible inconsistencies. In contrast, synthetic feedback is based on a simple decision derived from the current reward, ensuring more consistent results. Additionally, the authors suggest that labeled segments may not be well distributed in the state space, which could impact the effectiveness of human feedback.

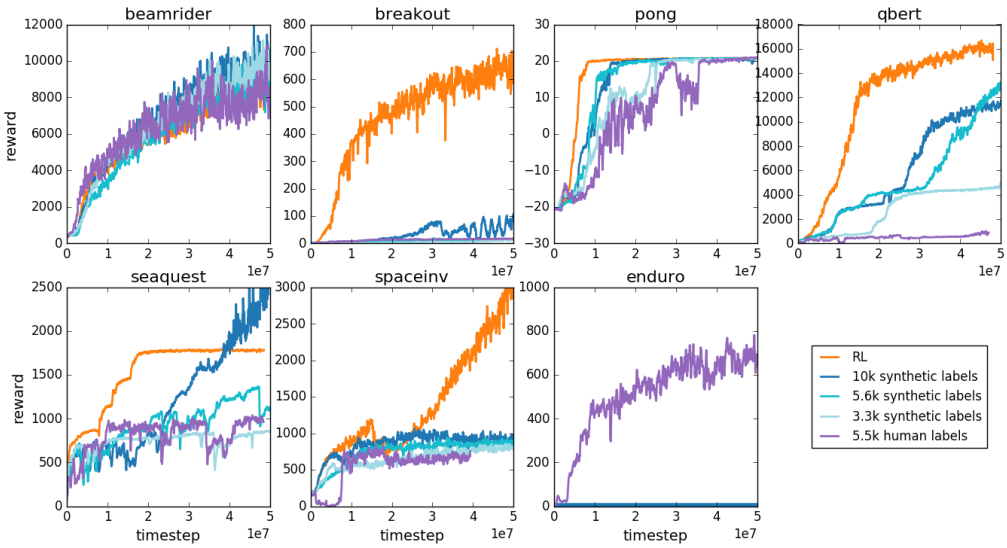


Figure 5.2: Results for Atari games from the original paper

On Figure 5.2, the goal is to approximate the RL reward function as close as possible. Thus, one can see that we have mitigated results.

5.1.3 Ablation studies

In this section, we summarize the ablation studies conducted by the authors. Within the context of artificial intelligence, an ablation study examines the importance of a specific component by removing it and analyzing its impact on overall performance. The authors explored various modifications to their system, evaluating how each change influenced the results.

1. **Random queries:** Instead of selecting queries based on disagreement among predictors (as described in section 4.4), queries are chosen at random.

2. **No ensemble:** Instead of using an ensemble of multiple predictors (as explained in section 4.5), only a single predictor is used. With these settings, it is impossible not to use random queries, since there is no more predictor to compute the variance and finally the disagreement.

3. **No online queries:** Human queries are only collected at the beginning of training rather than dynamically throughout the learning process.

4. **No regularization:** l_2 and dropout are not used anymore.

5. **No segments:** For robotics tasks only, the length of the segments are set at 1 (with the formalism introduced in section 4.1, it means $k = 1$).

6. **Target:** Instead of learning $\hat{r}$ through comparisons, we assume the presence of an oracle that provides the true total reward for a trajectory segment and train $\hat{r}$ by minimizing the mean squared error with respect to these total rewards.

The results obtained by the authors showed that online reward predictor training is crucial in order to capture the non-stationarity of the target. Regarding point 5, the authors observed that achieving similar results with single-frame comparisons (as opposed to short clips) required significantly more queries, which contradicts the goal of minimizing the number of human interactions. Additionally, evaluating individual frames took longer for humans, as isolated frames provide less context and are less intuitive to assess than short clips.

5.1.4 Novel behaviors

As mentioned in the introduction, the primary objective of the proposed system is to solve tasks where no explicit reward function is available. In the previous sections, we analyzed case studies where a predefined reward was used for comparison. However, the authors also demonstrated that their approach successfully enables agents to accomplish several challenging tasks without predefined rewards, including:

- The Hopper robot executing backflips.
- The Half-Cheetah robot moving while balancing on one leg.
- Maintaining position alongside other cars in Enduro.

5.2 Our Experiments

In this section, we detail the experiments we conducted in an attempt to reproduce the results. Given their accessibility and well-established benchmarks, we focused on Atari games rather than robotics problems. Among these, we specifically chose Pong, an iconic game in the field of reinforcement learning. Our implementation can be found on GitHub: Deep Reinforcement Learning From Human Preferences - M2DS.

Pong is a simplified simulation of a table tennis match, implemented in a module called Gym. Two players control paddles positioned on either side of the screen and must return a ball while preventing it from passing their side of the field. The ball follows a deterministic physics system: it bounces off the walls and changes trajectory based on the angle and point of impact on the paddles. The opponent (the left paddle) follows predefined rules to react to the ball's movement. The player controls the right paddle and must learn to defeat this AI by maximizing their score. A match lasts until one player reaches 21 points, setting an upper

limit on the game duration. This game is very useful to test RL algorithms due to its simple rules and its discrete yet dynamic environment.

The reward function implemented by Gym for Pong is designed to give +1 when the player successfully hits the ball and the opponent fails to return it (the ball passes the opponent's paddle). Conversely, the player receives -1 when they miss the ball and it passes their own paddle. For all other events, such as the ball bouncing off the walls or hitting the paddles, the reward is 0. This structure encourages the agent to maximize its score by successfully returning the ball and preventing the opponent from scoring.

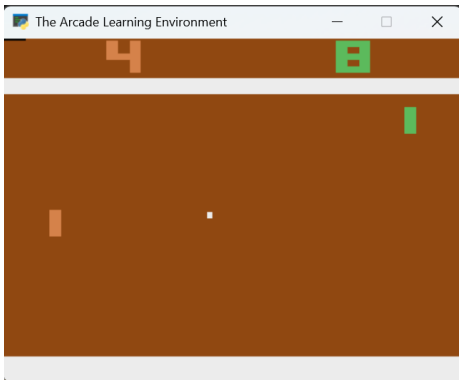


Figure 5.3: Illustration of the Pong game interface

We structured our experiments into two parts. First, we focused on implementing the A2C algorithm. Then, we explored incorporating human preferences to learn the reward function instead of using the one provided by the Gym environment.

5.2.1 Implementation of Advantage Actor-Critic

In this section, we detail our implementation of the A2C algorithm. The code for this section can be consulted here. We implemented the A2C algorithm using two methods:

Implementation from scratch (pong_A2C_train.py) This script uses the Stable-Baselines3 library and PyTorch to implement the A2C algorithm. The Pong environment returns images of the game as it progresses, so Convolutional Neural Networks (CNNs) are used to approximate both the Actor (policy) and Critic (state value) functions. We define the Actor and Critic classes, where the Actor generates a probability distribution for each possible action, and the Critic outputs a single value representing the estimated value of the current state.

After defining the classes, the `advantage_actor_critic` function is used to train the agent by maximizing the cumulative future rewards. This is done in a loop over multiple episodes. At the start of each episode, the environment is reset (a new game begins), and the values and rewards obtained at each stage are recorded. The loop continues for the specified number of epochs. During each iteration, the Critic predicts the current state's value, while the Actor predicts the probabilities of each action. An action is randomly chosen based on these predicted probabilities and added to a list. After executing the action in the environment, we store the new state's value and the received reward. This process continues until either the game ends (a player wins) or the maximum number of epochs is reached.

Once all epochs are completed, we compute the Q-value for each state and determine how much the agent's actual gain deviates from the Critic's initial estimate. This difference is then used to compute the Actor's loss, which decreases if the obtained reward exceeds the Critic's prediction and increases otherwise. The Critic's loss, on the other hand, is defined as the error between its prediction and the computed Q-value. Finally, the Actor and Critic CNNs are updated using the Adam optimizer, a process repeated at the end of each episode. Additionally, we record the total reward for every episode.

Regarding the algorithm, training follows a synchronous approach, meaning that the Actor and Critic are updated sequentially at the end of each episode. While this simplifies implementation, an asynchronous approach could potentially improve efficiency and convergence speed with parallel updates. The A3C model [8], for instance, suggests running multiple environments in parallel. This approach allows the agent to learn from a diverse set of experiences more rapidly, thereby accelerating the training process.
Unfortunately, we were unable to obtain conclusive results for this A2C implementation due to time and computational constraints. While we managed to run our algorithm for a limited number of episodes, it remains difficult to assess its performance. According to the authors' illustration Figure 5.2, the reward only starts to increase significantly after approximately one million timesteps. Reaching this threshold requires a substantial amount of training time, which becomes impractical without the use of a GPU.

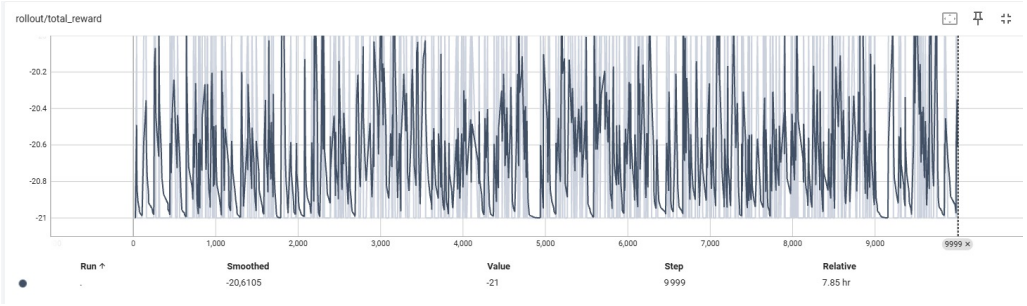


Figure 5.4: Evolution of our A2C model on 10 000 episodes processed in 8 hours. At the beginning of the process, the agent does bad decisions so we can make the hypothesis that it loses at least every 2 timesteps. With this hypothesis, we can then suppose that there is a maximum of 41 timesteps in each episode (as there is a maximum of 21 points for the winner). In 10 000 episodes, we can estimate that we only have 420 000 timesteps < 1 000 000 timesteps to see if the model improves at one point or not.

Using the Stable Baselines3 module "A2C" (pong_A2C_module.py) During our research, we discovered that the Stable Baselines library provides its own implementation of A2C. Therefore, we wanted to test it to verify if the A2C reward increased as shown in the authors' illustrations after approximately 1 million timesteps. Additionally, testing this model allowed us to introduce it into the human preferences framework, thereby simplifying the code.

We ran the algorithm for about 12 hours and managed to reach just over 5 million timesteps. With this, we observed that the reward began to improve after approximately 1 million timesteps, as indicated in the graphs of the Figure 5.2. Once we reached this result we focused on the integration of human preferences to the RL algorithm.

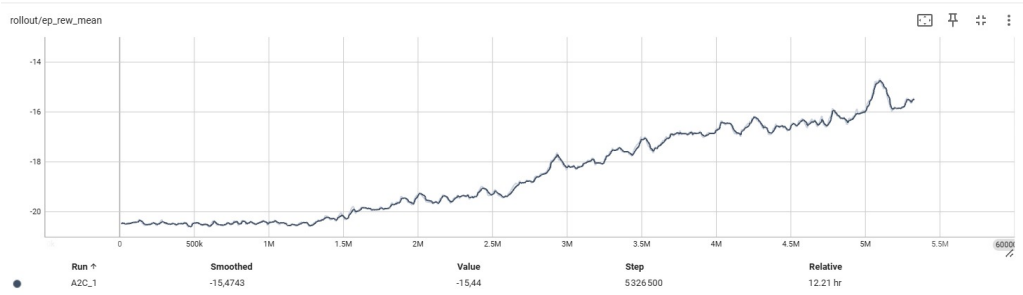


Figure 5.5: Evolution of the reward according to the episode using the function from the Stable_baselines3 module.

5.2.2 Implementation of Human preferences

In this section, we focused on training the reward using human preferences. To achieve this, we created a Gym wrapper called `HumanPreferencesEnvWrapper` to integrate preference collection and reward learning. To write this code, we drew inspiration from the code provided by mrahtz/learning-from-human-preferences and HumanCompatibleAI/learning-from-human-preference. These two repositories describe the implementation of the paper *Deep Reinforcement Learning from Human Preferences* for various Atari games, including Pong. We only took inspiration from the structure of what they wrote, without copying, and created our own classes and scripts. We also developed a simpler version to ensure we fully understood each step we implemented. Our code concerning this part can be found here. To test our code, you can run the script `drlhp_test_A2C_video.py` which uses classes in `drlhp_train_A2C_video.py`

HumanPreferencesEnvWrapper When an action is executed, the wrapper sends this action to the Gym environment and retrieves the new observation, the immediate reward, and the information indicating whether the episode has ended (`done`). This observation is then added to an current segment, which groups a sequence of images and rewards. Once the segment reaches a predefined length, it is stored in `self.segments` for future use. As soon as two or more segments are available, the user interface (`PrefInterface`) selects a pair and prompts the user to express a preference between the two. If a preference is provided, the `RewardPredictor` model is trained to adjust its reward predictions based on the user's choices, allowing the agent's learning process to be shaped by human preferences rather than a fixed game reward. Midway through training, we transition from using the environment's default reward signal—which is implemented in Gym—to the learned reward function, which is shaped by human preferences. This switch is performed by calling `switch_to_predicted_reward()`, which modifies the environment so that the agent no longer relies on the original game rewards but instead receives rewards predicted by the `RewardPredictor` model. At this stage, the agent's learning process is entirely driven by the reward model that has been trained using human feedback, allowing it to optimize its behavior based on what humans find preferable, rather than the built-in game scoring system.

RewardPredictor is the class containing the neural network supposed to learn a reward function based on human preferences. The network consists of three fully connected layers with ReLU activations, progressively reducing the input dimension to a single output value, which represents the predicted reward for a given segment. During training, the model takes

two segments (`s1` and `s2`) and averages their frames to create a single representative input for each segment. These processed inputs are then passed through the network to compute their respective predicted rewards (`r1` and `r2`). The model is trained using a preference-based loss function, which applies a sigmoid transformation to the difference ($r_1 - r_2$) and compares it to the user's expressed preference using binary cross-entropy loss (`BCEWithLogitsLoss`). The optimizer (`Adam`) updates the network's weights based on this loss, gradually refining the reward function to align with human preferences.

PrefInterface is responsible for collecting and managing human preferences in order to train the reward model. It maintains a buffer of segments, where each segment consists of a sequence of observations and rewards. When at least two segments are available, the interface selects a pair of segments that have not yet been compared and prompts the user to express a preference between them. To facilitate this, the interface displays both segments side by side using OpenCV (`cv2.imshow()`), allowing the user to visually compare them. The user can then indicate their preference by selecting `L` (left segment), `R` (right segment), `E` (equal preference), or `Q` (skip comparison). The selected preference is stored and used to train the `RewardPredictor`. To avoid redundant comparisons, the interface keeps track of tested segment pairs using a hash-based system, ensuring that each segment is only compared with others in a meaningful way.

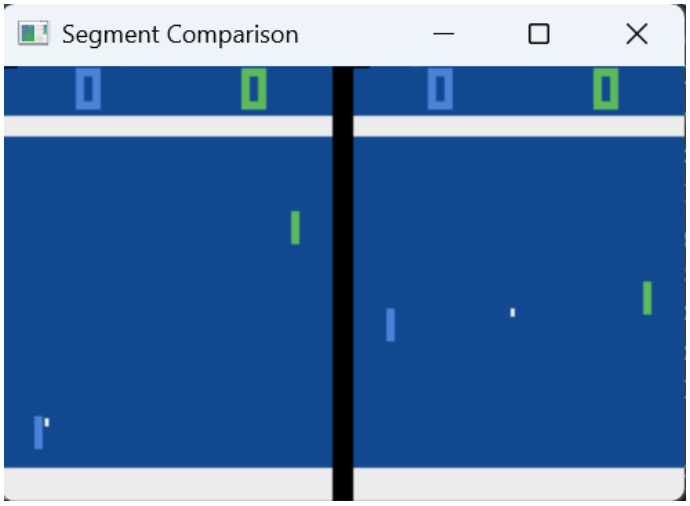


Figure 5.6: Illustration of the Window showing the two segments to compare

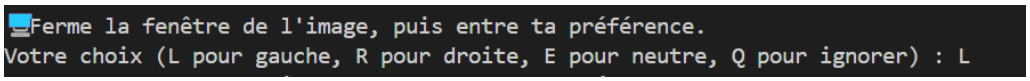


Figure 5.7: Illustration of the Terminal, asking the user to make a choice

Potential Improvements

There are several areas where our implementation could be improved to better align with the approach described in the paper:

- First, the selection of segment pairs for comparison is currently done randomly, whereas the paper suggests using an uncertainty-based approach to prioritize the most informative

queries, as described in Chapter 4.4: they train an ensemble of reward predictors and measure disagreement between them to estimate uncertainty. Segment pairs with the highest uncertainty are prioritized for human feedback, reducing the number of queries needed while maximizing learning efficiency. Implementing this strategy in our pipeline could significantly improve query efficiency.

- The paper does not request user preferences at a fixed rate but instead decreases the query frequency over time using an "annealing schedule" that gradually reduces the number of queries as the model becomes more confident. Early in training, more feedback is requested, while later, only the most uncertain comparisons are prioritized. Incorporating such a schedule in our implementation would prevent unnecessary queries and improve sample efficiency.

- The preprocessing of observations could be enhanced by using CNNs instead of simply flattening the input, as CNNs better capture spatial features in environments with visual data, such as Atari games.

- The training of the reward model in our implementation is sequential, waiting for a sufficient number of segments before updating. The paper, however, emphasizes an asynchronous approach where the reward model and policy are trained in parallel. Implementing such an asynchronous process could improve the efficiency and responsiveness of our learning pipeline.

- We do not currently use the exact same loss function as described in the paper, which relies on a cross-entropy formulation over preference probabilities. Integrating this loss could further improve the alignment of our approach with the original methodology.

Our implementation differs from the code we took as reference in several key aspects. As the authors did, they use an ensemble of reward predictors to estimate uncertainty, they prioritize uncertain segment pairs for user queries and train the policy and reward model asynchronously using multiprocessing queues. To enhance the robustness and scalability of our model, we could integrate some of these ideas into our code, adopting a modular and distributed approach where preference collection and model training are handled separately through multiple processes. This could involve implementing a reward predictor ensemble (RewardPredictorEnsemble), a preference database (PrefDB, PrefBuffer), and a multi-process system (SubprocVecEnv, Queue, Process) to optimize learning efficiency.

Results of this experiment

Regarding the results obtained on our model's performance, we encountered some difficulties. Indeed, if we refer to Figure 5.2 showing the Atari results and look at the Pong game, we see that the authors achieve the same reward as the classic reward after around 40 million timesteps and with 5.5k human labels. However, without a GPU, it is difficult to reach 40 million timesteps in a reasonable amount of time (for the A2C algorithm, we succeded to reach around 6 millions timesteps in 12 hours !!!). Additionally, we have seen our implementation differs from the one in the article, and we request significantly more human preferences in our script than in the original work, making it very complex to properly test our algorithm. As a result, we do not know whether our model performs correctly. However, we have verified that the preference interface works, and building this code has allowed us to gain a much deeper understanding of how the algorithm works, making this experience highly instructive.

An interesting approach, present both in the reference code and the paper, is the use of synthetic preferences. This could help reduce the burden on human annotators by avoiding the need for 5,500 manual responses during training. In the reference code, synthetic preferences are generated based on the predicted reward sums of the segments, which come from a pre-trained reward prediction model. This model is trained beforehand on specific data and then used to estimate the rewards for each segment. Preferences are assigned in this way: if the sum of predicted rewards for one segment is higher than for another, it is preferred; if the rewards are equal, the preference remains neutral. Implementing synthetic preferences in our work could make the training process significantly more feasible, allowing us to test our model more effectively before involving human input.

Chapter 6

Discussion and conclusions

In this report, we analyzed the paper *Deep Reinforcement Learning from Human Preferences* [1], providing an overview of the key related works to frame the significance of the research. We also introduced the proposed methodologies, offering additional insights to help comprehension. After reviewing the authors' experiments, we focused on implementing their approach, specifically creating an interactive interface to train an agent playing the Pong game using human feedback. Despite computational and time constraints, we successfully developed a prototype, but further improvements are needed to replicate the results from the original paper.

Overall, the approach presented in the paper demonstrates that algorithms trained with minimal human input can be both effective and time-efficient. By reducing the number of queries made to humans and making the system accessible to non-experts, it highlights the potential of learning from human preferences while relying on an explicit reward function. Future research could further refine the efficiency of these learning methods, making them even more practical for real-world applications. In the long run, the aim is to develop reinforcement learning systems capable of learning from human preferences as easily as they learn from predefined reward signals. This advancement could revolutionize the way we design, train, and interact with AI, opening new possibilities for more intuitive and adaptive systems.

Bibliography

[1] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[2] Jimmy Secretan and Nicholas Beato. Picbreeder: evolving pictures collaboratively online. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008.

[3] Eric Brochu, Tyson Brochu, and Nando de Freitas. A bayesian interactive optimization approach to procedural animation design. In *Symposium on Computer Animation*, 2010.

[4] Riad Akrour, Marc Schoenauer, and Michèle Sebag. April: Active preference-learning based reinforcement learning, 2012.

[5] Aaron Wilson. A bayesian approach for policy learning from trajectory preference queries. 12 2012.

[6] Marc Schoenauer, Riad Akrour, Michele Sebag, and Jean-Christophe Souplet. Programming by feedback. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1503–1511, Bejing, China, 22–24 Jun 2014. PMLR.

[7] Johannes Fürnkranz, Eyke Hüllermeier, Weiwei Cheng, and Sang-Hyeun Park. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156, 2012.

[8] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[9] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[11] Shaofeng Zou, Tengyu Xu, and Yingbin Liang. Finite-sample analysis for sarsa with linear function approximation, 2019.

[12] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[13] Roger N. Shepard. Stimulus and response generalization: tests of a model relating generalization to distance in psychological space. *Journal of experimental psychology*, 55 6:509–23, 1958.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[15] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint*, arXiv:1606.01540, 2016.

List of Figures

4.1 Overall architecture proposed 9

5.1 Results for simulated robotics tasks from the original paper 15
5.2 Results for Atari games from the original paper 16
5.3 Illustration of the Pong game interface 18
5.4 Evolution of our A2C model on 10 000 episodes processed in 8 hours. At the beginning of the process, the agent does bad decisions so we can make the hypothesis that it loses at least every 2 timesteps. With this hypothesis, we can then suppose that there is a maximum of 41 timesteps in each episode (as there is a maximum of 21 points for the winner). In 10 000 episodes, we can estimate that we only have 420 000 timesteps < 1 000 000 timesteps to see if the model improves at one point or not. 19
5.5 Evolution of the reward according to the episode using the function from the Stable_baselines3 module. 20
5.6 Illustration of the Window showing the two segments to compare 21
5.7 Illustration of the Terminal, asking the user to make a choice 21