Introduction to Reinforcement Learning

Master 2 Data Science

Deep Reinforcement Learning from Human Preferences

Ivo Bonetti and Tom Salembien

Code et implementation GitHub link

Teacher :
Erwan LE PENNEC

Contents

1 Introduction

Nowadays, reinforcement learning has been able to solve large tasks. In order to do so, the different algorithms use a very specified reward function. However, in the real life, tasks, challenges, problems are not always specific nor well-defined. Therefore, it can be really difficult to build a reward function to optimize such problems. This is the case mostly for daily tasks such as cooking. The objective of deep reinforcement learning is to overcome the reward function definition issue in order to solve these complex problems.

Furthermore, here the second objective is to have a reinforcement learning algorithm that satisfies human preferences. Otherwise, it would remain a classic reinforcement learning algorithm. Indeed, it could be possible to design a reward function capturing the behavior, the task that we want it to accomplish. However, it would be a simple reward optimization and would not take into account human preferences.

To satisfy human preferences, two ways are possible. The first one by using an inverse reinforcement learning algorithm as proposed by Ng and Russel in 2000 [8]. The idea is extracting the reward function. This can be done if demonstrations of the desired task are available. It can be a good solution because according to Ng and Russel, "the reward function rather than the guideline, is the most concise, robust and transferable definition of the task". Indeed, the reward function can help us put a number to qualify actions that are made. Guidelines are more blurry. As mentioned before, inverse reinforcement learning can be used with demonstrations. The issue stands in this point. As a matter of fact, demonstrations result in a list of behaviors that can be provided by humans, experts in their domain which is a problem because experts are not always available. Therefore, it would be difficult to show the proper behavior when humans can not demonstrate it.

This is where the second approach described by Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg and Dario Amodei comes in. A human can provide feedback on the behavior of the system. This feedback will be considered as the reward function that the algorithm is trying to optimize. Moreover, this feedback is also the preferences satisfaction that is aimed. However, it is quite expensive. It is estimated at hundreds or thousands hours experience in order to learn properly. Thus, the objective of the algorithm is to solve tasks with a specific behavior given but that has not always been demonstrated. The feedback used can be given by non-experts on large problems without requiring too much time.

To do so, the human will compare short video clips and give his feedback. These clips are provided by the algorithm after observing the environment. The feedback given by the human will give a score to the reward predictor. Then, this predicted reward trains the algorithm.

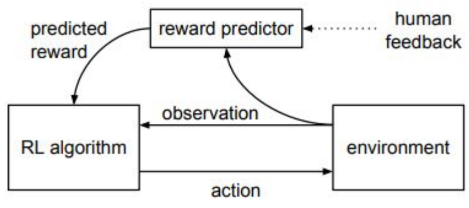


Figure 1: Schematic illustration of the approach

2 How does it work ?

First, we consider that an agent interacting with an environment receives at each time step t an observation $o_t \in \mathcal{O}$. Then, the agent replies with an action noted $a_t \in \mathcal{A}$. Where $\mathcal{O}$ is the set of observations and $\mathcal{A}$ is the set of actions. This is where the biggest difference with traditional reinforcement learning algorithms occurs. Indeed, in traditional reinforcement learning, the environment gives a reward r_t at each time step in order to be maximized by the agent over time. However, our goal is to overcome this reward function by satisfying human preferences. Therefore, a different method will be used.

To satisfy the human preferences, we have explained that short video clips will be used for comparisons. As a matter of fact, the agent gives two trajectory segments. A trajectory segment σ is a sequence of couples of observations and actions. It is defined as follows:

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

This trajectory segment represents the video clips that are compared by a human showing its preference by telling which trajectory is better according to him. In the meantime, the agent must make as few queries as possible by producing trajectories that are most likely to be preferred by the human.

Hence, we have two ways to evaluate the model. First, quantitatively. This is based on the human preference according to the trajectories proposed by the agent.

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

Which can be seen as:

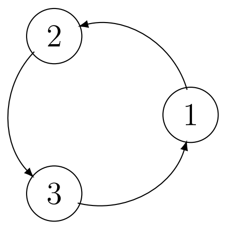
$$\sum_{i=0}^{k} r(o_i^1, a_i^1) > \sum_{i=0}^{k} r(o_i^2, a_i^2)$$

This can be seen as a reward function (r) where the objective is the same as in traditional reinforcement learning, maximizing the reward. In a perfect world, the agent will have the same score as if a reward function was given explicitly. Then, the model can be evaluated qualitatively. This is the case when we do not have any reward function. Thus, to evaluate the model, the human evaluates the behavior of the agent by considering on the agent's performance attempting to fulfill the goal. The goal can be given in natural language.

The model used to satisfy the human preferences is a asynchronous process. Indeed, when we look at the Figure 1, we are able to see the asynchronous process. The reward predictor is trained at a different time step from the comparisons of the trajectory segments. To be more specific, three processes running asynchronously help to achieve the goal defined. These processes, updating the networks of the deep neural networks build, are:

1. Policy π interacting with the environment, and producing a set of trajectories $\tau^1, ..., \tau^i$. The parameters of π are updated through traditional RL algorithm in order to maximize the sum of rewards.
2. Selection of a pair of segments (σ^1, σ^2) from the trajectories to be compared by a human.
3. Parameters of $\hat{r}$ are optimized via the fitting of the comparisons made by humans.

The flow is the following:



2.1 Optimizing a policy

Foremost, the objective is to optimize a policy. The main issue comes from the fact that the reward function $\hat{r}$ may be non-stationary. Therefore, the method used must be robust to counter it. A solution is the policy gradient methods. Two different algorithms are used here. Advantage actor-critic and Trust Region Policy Optimization (TRPO). The Advantage Actor-Critic method is a variant of the Actor-Critic methods. Hence, we may recognize our way of evaluation with the Actor that controls the behavior of the agent and the Critic measuring the quality of the action.

2.1.1 Policy Gradient

In order to have a better understanding of the two algorithms used in the paper, a reminder on how Policy gradient [3] methods work is useful.

First, we need to define $\tau = (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T)$; it is a state-action pair trajectory of T steps. In addition, we have the reward function $R(s_t, a_t)$ which corresponds to the gain obtained for an action a_t during the state at the same time step t. Besides, π_θ is the policy. Thus, on the whole trajectory, we can write: $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$. With this reward, the objective will be to maximize the expected reward such as below:

$$max_\theta \mathbf{E}_{\pi_\theta}[R(\tau)]$$

To fulfill the goal previously mentioned, it is important to understand what it means through the formula given. We can reformulate the objective by saying that the aim is to find the best parameters that will allow us to obtain the policy which, then, will maximize the expected reward.
Now that we have redefined what we want, we need to set up a method to reach our goal. Gradient ascent method can be useful in this case to find the best parameters.

Therefore, the algorithm to find θ would be:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbf{E}_{\pi_\theta}[R(\tau)]$$

With $\nabla_\theta \mathbf{E}_{\pi_\theta}[R(\tau)]$ the gradient of our final objective with respect to the parameter we are trying to optimize and α the learning rate.
We will now decompose $\nabla_\theta \mathbf{E}_{\pi_\theta}[R(\tau)]$ to go deeper in its understanding.
On the one hand, we have:

$$\begin{aligned} \nabla_\theta \mathbf{E}_{\pi_\theta}[R(\tau)] &= \nabla_\theta \sum_\tau P(\tau|\theta)R(\tau) \\ &= \sum_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\ &= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta)R(\tau) \\ &= \sum_\tau P(\tau|\theta)\nabla_\theta log P(\tau|\theta)R(\tau) \\ &= \mathbf{E}_{\pi_\theta}(\nabla_\theta log P(\tau|\theta)R(\tau)) \end{aligned} \tag{1}$$

On the other hand:

$$P(\tau|\theta) = p(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus, the gradient of log probability is:

$$\begin{aligned} \nabla_\theta log P(\tau|\theta) &= \nabla_\theta (log p(s_0) + \sum_{t=0}^{T-1}(log p(s_{t+1}|s_t, a_t) + log \pi_\theta(a_t|s_t)) \\ &= \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) \end{aligned} \tag{2}$$

This is true because:

$$\pi_\theta(\tau)\nabla_\theta log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

Finally, we have:

$$\nabla_\theta E_{\pi_\theta} T(\tau) = E_{\pi_\theta}(\sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t)R(\tau)$$

In practice, we can estimate this expectation using Monte-Carlo sampling:

$$\nabla_\theta E_{\pi_\theta} T(\tau) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t^i|s_t^i)R(\tau^i), \text{with N, the number of trajectories for the update}$$

Therefore, we can combine with the gradient ascent method which gives the following algorithm:

1. Sample τ^i
2. $\nabla_\theta E_{\pi_\theta} T(\tau) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t^i|s_t^i)R(\tau^i)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta \mathbf{E}_{\pi_\theta}[R(\tau)]$

2.1.2 Actor-Critic Architecture

This architecture is a hybrid approach combining a value-based and policy-based methods. The previous combination will help stabilize training and reduce variance which is a big issue with policy gradient method. The Actor-Critic architecture [1] introduces the Q value as a result of policy gradient decomposition. Here is a simple example:
We have the policy gradient below:

$$\nabla_\theta J(\theta) = \mathbf{E}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)Gt], \text{where } s_t \text{ is the state and } G_t \text{ the total return}$$

Then, we can decompose the expectation:

$$\nabla_\theta J(\theta) = \mathbf{E}_{s_0,a_0,\ldots,s_t,a_t}[\sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)]\mathbf{E}_{r_{t+1},s_{t+1},\ldots,r_T,s_T}[G_t]$$

Finally, we can recognize Q:

$$\mathbf{E}_{r_{t+1},s_{t+1},\ldots,r_T,s_T}[G_t] = Q(s_t, a_t)$$

To explain with words:

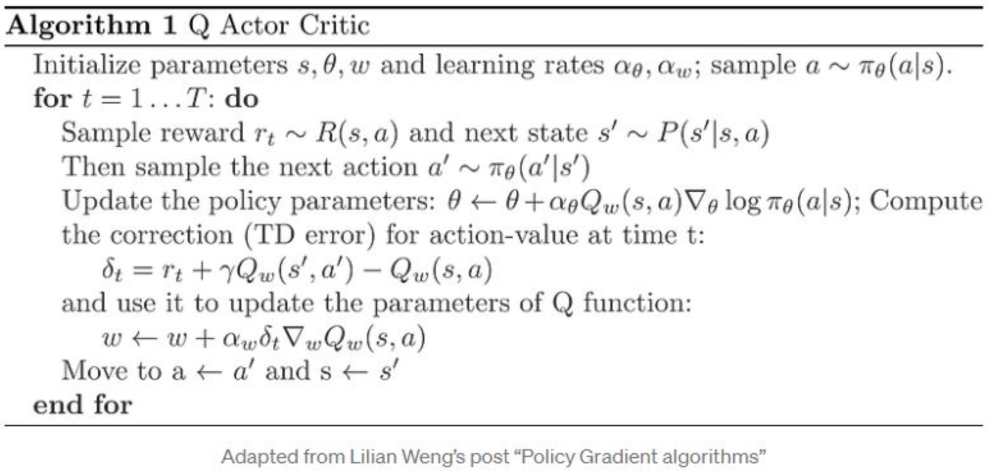
1. The ***Critic*** estimates the value function. Here, it is represented by the *Q-value*

2. The ***Actor*** which updates the policy distribution as suggested by the ***Critic***

Here is a summary of the expression of the functions metionned:

Figure 2: Gradient function from policy gradient

To parameterized the two functions, neural networks can be used.
Below, we will give the pseudo-code for Q-Actor-Critic algorithm (same idea for Advantage-Actor-Critic). The w represents the parametrization via a neural network: In order to have a better representation of what is Actor-Critic method, here



```
Algorithm 1 Q Actor Critic
  Initialize parameters $s, \theta, w$ and learning rates $\alpha_\theta, \alpha_w$; sample $a \sim \pi_\theta(a|s)$.
  for $t = 1 \dots T$: do
    Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
    Then sample the next action $a' \sim \pi_\theta(a'|s')$
    Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute
    the correction (TD error) for action-value at time t:
        $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
    and use it to update the parameters of Q function:
        $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    Move to a $\leftarrow a'$ and s $\leftarrow s'$
  end for
```

Adapted from Lilian Weng's post "Policy Gradient algorithms"

Figure 3: Q-Actor-Critic pseudo code

is an example:
The ***Actor*** can be viewed as a player of a video game, and the ***Critic*** as a friend who reviews your in-game actions. Depending on what your friend advives you to do, you apply the advice to have a better score in the game. This is the same here in Reinforcement Learning. In the case of this paper, the method implemented is the Advantage Actor-critic(A2C).

2.1.3 Advantage Actor-Critic Architecture (A2C)

In order to have the method called Advantage Actor-Critic [2], we have to transform the Q-value into the Advantage value. To do so, we make the difference between the *Q-value* and the average value of the state where in (V(s_t, a_t)) The Advantage value is shown below.

$$A(s,a) = \underbrace{Q(s,a)}_{\text{q value for action a in state s}} - \underbrace{V(s)}_{\text{average value of that state}}$$

Figure 4: Advantage value expression in A2C

This is interesting because depending on the value of A, the gradient will be pushed in the good direction ($A(s,a) < 0$) or in the opposite direction ($A(s,a) > 0$).

Finally, the method used in the paper is a derived form from the A2C. Indeed, we have seen that the process is asynchronous. This asynchronism can be brought in the Actor-Critic method which leads to the Asynchronous Advantage Actor-Critic method also known as A3C. The agents (or workers) are trained in parallel and update periodically a global network, which holds shared parameters. The updates are not happening simultaneously and that's where the asynchronous comes from. After each update, the agents resets their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again.

We see that the information flows not only from the agents to the global network but also between agents as each agent resets his weights by the global network, which has the information of all the other agents.

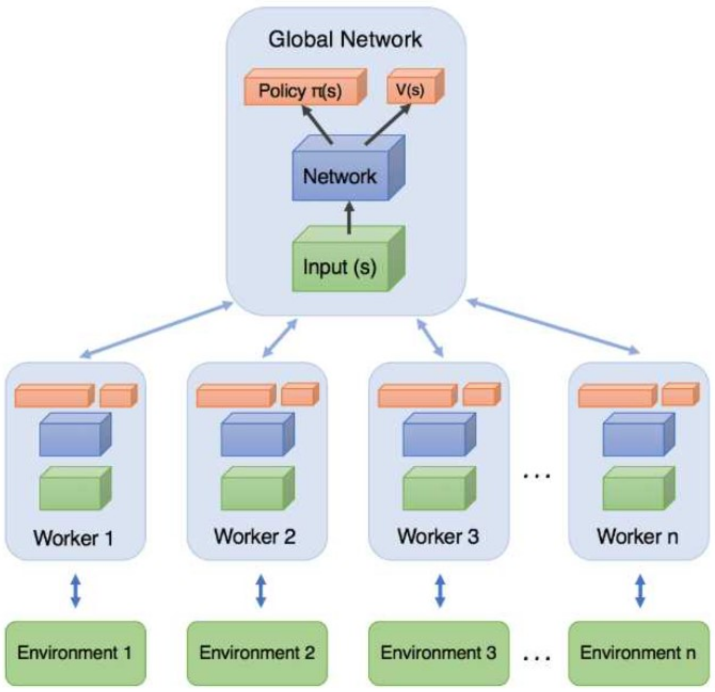


Figure 5: A3C (Asynchronous Advantage Actor-Critic) model

2.1.4 Trust Region Policy Optimization (TRPO)

On the other hand, the TRPO algorithm [9] [4] is interesting because it introduces also an advantage value.
If we write the expected discounted return (the performance metric):

$$J_\pi = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathbf{R}_k \right]$$

With the advantage, $a_\pi(x,u) = q_\pi(x,u) - v_\pi(x)$, the updated policy is written as follow neglecting the state distribution change due to a policy update (local approximation $p^\pi(x) = p^{\tilde{\pi}}(x)$),

$$\mathcal{L}_\pi(\tilde{\pi}) = J_\pi + \int_{\mathcal{X}} p^\pi(x) \int_{\mathcal{U}} \tilde{\pi}(u|x) a_\pi(x,u)$$

However, we do not know how much the actual stochastic policy will change while moving through the parameter space. Hence, we do not have a good decision basis to choose the policy gradient step size.

Even if the TRPO is not quite different from natural policy gradient methods, three main improvements make the TRPO significantly better. As a matter of fact, the TRPO aims to verify whether the updates proposed improve our algorithm. To obtain such result, TRPO implements

- a conjugate gradient method, line search based on Divergence of Kullback-Leibler as we want a metric describing how much a policy is changed in the action space when updating the policy in the parameter space written $D_{KL}(\pi_\theta)||\pi_\theta + \delta_\theta)$

- an improvement check ($\mathcal{L}$, 0). In the method proposed to satisfy human preferences, the hyperparameter controlling the exploration was adjusted manually to ensure adequate exploration.

These two algorithms are useful when it comes to simulation of robotic tasks. Besides, the rewards produces are normalized (zero-mean and constant standard deviation) because the position of the rewards are not known.
Then, we have the preference elicitation. The short video clips (1 or 2 seconds long) are given to a human that must indicate if a segment is better than the other, equally good or if he is unable to compare them. Depending on the judgement, a triple containing the two segments and a distribution μ translate the human judgement. If a segment is better than the other, the distribution μ will put its mass on the segment selected. If the segments are equal, then the distribution is uniform. If it is incomparable, the judgement is not included. Furthermore, this feedback elicitation follows the work of Wilson (2012) [10] on Bayesian approach to determine which segments are more likely to be chosen as good.

2.2 Preference elicitation, Bayesian model (2012)

To go deeper into the preference elicitation model, we will look at the work of Wilson, Fern and Tadepalli with their paper **A Bayesian Approach for Policy Learning from Trajectory Preference Queries**.

Their model is based in the framework of Markov Decision Processes (MDP). This MDP produces sequence of state-action pairs defined as trajectories, which can be

of a length K. Starting from the state point s, *Trajectory Preference Queries* (TPQ) are given to an expert which assess them. A TPQ is the pair of two trajectories that can be noted (ξ_i,ξ_j). The result y given by the expert creates a training data tuple. This training data will be used then to learn a policy.
The expert evaluation function is noted $f(.)$. Thus, $y = 1(f(\xi_i) > f(\xi_j))$. We can assume that expert's evaluation function is a function. Besides, the latent target policy is noted θ^*.

2.2.1 Bayesian model and inference

It is important to define first the distribution of the expert's response. It can be described as such : $P(y|(\xi_i, \xi_j), \theta^*)P(\theta^*)$. We can recognize the prior over the latent expert policy ($P(\theta^*)$) and the response distribution(($P(y|(\xi_i, \xi_j), \theta^*)$).

Response distribution:

$$P(y = 1|(\xi_i, \xi_j), \theta^*) = \int_{-\infty}^{+\infty} 1(f^*(\xi_i, \xi_j, \theta^*) > \epsilon)N(\epsilon|0, \sigma_r^2)\, \mathrm{d}\epsilon$$

This is the explanation of the response distribution where the indicator reflects the expert response (if the trajectory i is preferred rather than the trajectory j) and where ϵ $N(0, \sigma_r^2$. The function f^* shows how well the latent policy θ^* matches with ξ_i relatively to ξ_j. It is also possible to see the response distribution as the cumulative distribution function of the normal distribution. This formulation permits mistake from the expert.

Evaluation function:
The objective of the function is to measure the distance between the trajectories of the query and the trajectories generated by the latent target policy. Therefore, f can be written such as below because it measures the difference between the trajectories of a query:

$$f(\xi_i, \xi_j) = \sum_{t=0}^{K} k([s_{i,t}, a_{i,t}], [s_{j,t}, a_{j,t}])$$

Where $[s_{i,t}, a_{i,t}]$, $[s_{j,t}, a_{j,t}]$ is the pair of state-action pair respective to the trajectories i and j. In the paper, k is a simple distance; $k([s, a], [s', a']) = ||s - s'|| + ||a - a'||$. Then, we compare the distance between a trajectory given by a query and a trajectory generated by the policy θ^*. The measure is:

$$d(\xi_i, \theta^*) = \mathbf{E}[f(\xi_i, \xi_j)]$$

By combining the two functions, we are able to compute the difference of the distances of each trajectory given by the query relatively to the trajectory of the policy.

$$f^*(\xi_i, \xi_j, \theta^*) = d(\xi_j, \theta^*) - d(\xi_i, \theta^*)$$

The larger is f^*, the stronger the preference is on ξ_i trajectory as the distance between ξ_i and θ^* is low.

Finally, the posterior distribution can be approximated using a Hybrid Monte Carlo algorithm. This method is used for policy selection. Indeed, the authors decided to select a policy that generates samples and then pick the one maximizing the energy function $(log(P(D|\theta)P(\theta)))$.

2.2.2 Active Query Selection

Active query selection is important because the queries that are made must be the most useful as possible in order to fulfill the objective; which is learning the target policy. However, it can be difficult in practice due to high dimensional space. Two approaches can be considered.

Query by disagreement:
The main idea behind this approach is close to the query-by-committee. The query-by-committee generates unlabeled examples from a distribution and a pair of classifiers from the posterior. If the classifiers disagree on the label then a query is made. Here it is quite the same. Policies are drawn from a posterior distribution $P(\theta^*|D)$ where D is the set of training data. Then, if we observe a disagreement between the policies on the state observed, a query is made on the trajectories that have been generated by each policy. It can be measured via the following function g, $g = \int_{(\xi_i,\xi_j)} P(\xi_i|\theta_i, s_0, K)P(\xi_j|\theta_j, s_0, K)f(\xi_i, \xi_j)$. Then, it can be estimated through the sampling of a set of K-length trajectories taken from each policy.
If, the measure is above a defined threshold, a trajectory preference query is generated to be assessed by the expert. The expert assess the K steps of the trajectory.
This method is quite interesting because it creates TPQs when the trajectories ξ_i and ξ_j are significantly different. A benefit that can be observed then is that because the trajectories are quite different, the assessment can be easier. Therefore, it is less

prone to misjudgement from the expert.

Expected Belief Change:
This approach is dissimilar from the previous one. Indeed, the approach is based on Bayesian heuristic strategy. More specifically, it uses the Kullback-Leibler Divergence to measure quality of the experiments. This leads to a symmetric measure of the difference in the probability instead of expected differences in code lengths. Moreover, the computational cost is greater than in the previous approach. Nevertheless, its performance is better.
This strategy works with the use of variational distance between posterior based on current data D and the updated data $D \cup (\xi_i, \xi_j, y)$; with $V(P(\theta|D)||P(\theta|D \cup (\xi_i, \xi_j, y)))$ which is equal to:

$$V(P(\theta|D)||P(\theta|D \cup (\xi_i, \xi_j, y))) = \int |P(\theta|D) - P(\theta|D \cup (\xi_i, \xi_j, y))| \, \mathrm{d}\theta$$

Then, the agent computes the expectation of the variational distance,

$$H(d) = \sum_{y \in 0,1} P(y|\xi_i, \xi_j, D) V(P(\theta|D)||P(\theta|D \cup (\xi_i, \xi_j, y)))$$

Where, $P(y|\xi_i, \xi_j, D) = \int P(y|\xi_i, \xi_j, \theta^*) P(\theta^*|D) \, \mathrm{d}\theta^*$ is the predictive distribution and is estimated using posterior samples.
Thus, the variational distance is also estimated using a Monte-Carlo method to decide whether a query should be made.

2.3 Fitting the reward function

Finally, we have to fit the reward function estimate $\hat{r}$ (as a preference-predictor) and select the queries. To do so, the probability that a segment is preferred rather than the other one is defined as follows:

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

Besides, to optimize the reward function, we want to minimize the cross-entropy loss such as below:

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

The idea behind the use of these functions is based on Elo ranking in chess but by estimating the probability of a player winning, the approach used here is different by using the difference between predicted reward.

Some interesting points are the fact that the predictors are trained on the triples made by the human judgement, and then the estimate of the reward function is the average of the normalized predictors. Moreover, l2 regularization has been used to keep the validation loss between 1.1 and 1.5 times the training loss. Dropout could have been applied to obtain similar results. Finally, as human are inclined to make errors, to make sure that the function decay to 0. This is combined to the way of selecting the queries that will be presented to the human. Using a reward predictor, the agent is able to sample a large number of couples of trajectories. Then, it approximates the variance of each pair and give back the ones with the greatest variance.

3 Experimental Results

The results of the experiments come from different tasks where the true reward was not observable. The agent learns how to fulfill the task by asking the human which segment is better between the two proposed. In the meantime, the number of queries must be reduced to the minimum to decrease the amount of time required to fulfill the objective. To make comparisons, experiments are run using a synthetic oracle which reflect the reward of the task. These comparisons are made with a traditional Reinforcement Learning algorithm which was given the true reward function. The objective is to have the same results without the reward than with.

3.1 OpenAI Gym

OpenAI Gym [5] is a great open-source Python library for working and developing reinforcement learning algorithms. Before Gym existed, researchers faced the problem of unavailability of standard environments which they could use for development rapid prototyping of their algorithms. The advantages of this library is that it makes reinforcement learning a more practical and implementable advancement/alternative to traditional machine learning methods. Tanks to standard API that allows communication between learning algorithms and classic environments (as well as a standard set of environments compliant), it is now possible to compare the performance of reinforcement learning algorithms on the same standard environments.

3.2 Mujoco

MuJoCo (Multi-Joint dynamics with Contact) is a repository from DeepMind which aims to facilitate research and development in robotics, biomechanics, graphics and many areas. MuJoCo provides many physics engines to simulates fast and accurate articulated structures interacting with their environment. It is also possible to render interactive visualization with OpenGL GUI.

For the simulated robotics tasks, the model they used is explained in details with hyperparameters tuning, number of trajectories compared.. in the annexe A that contains Experimental Details of the article [6]). We won't reexplain them as there are only parametrization, the agent used here is TRPO.

3.2.1 The results

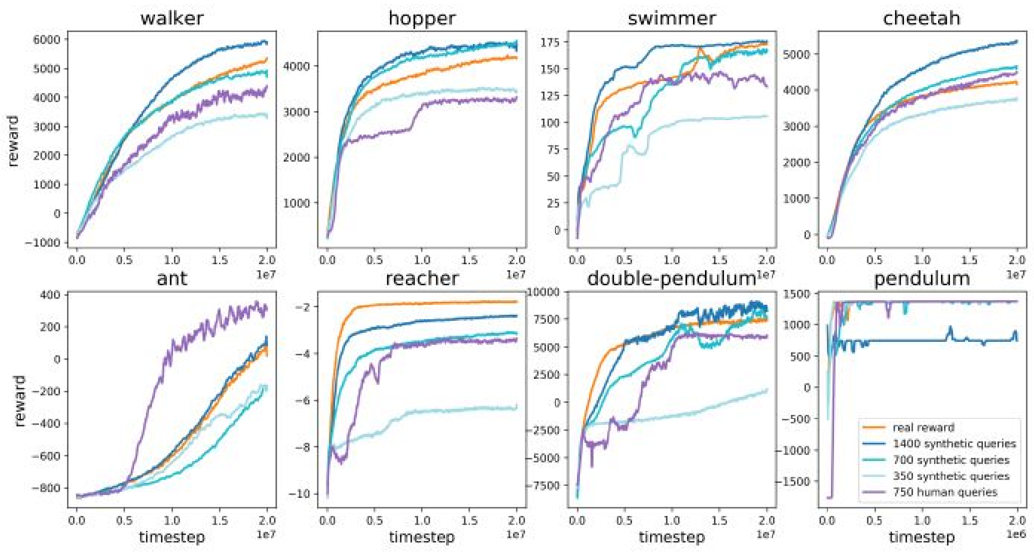


Figure 6: Results on MuJoCo simulated robotics

The results above are quite interesting. Using 700 synthetic labels, or queries, the algorithm tend to reach the performance of the reinforcement learning algorithm using the true reward. Besides, training with the knowledge of the reward function tends to have higher instability and variance. Having the double of synthetic labels does not improve the performance by two. The results are only slightly better than using the real reward. When it comes to real human feedback, it is less effective than synthetic feedback except on the simulation of ants where human feedback outperformed even the real reward algorithm. This can be explained by some instructions given to the people giving the feedback which was useful for reward shaping.

3.3 Atari

Atari is a console that contains some classic arcade games with very simple rules (for humans). Open AI Gym contains a lot of atari games environments that we will be tested in this article.

3.3.1 The model

In this article, the authors proposed to use "no-op actions" for the first 30 actions. Indeed, these no ops actions allow the player to become familiar with the game before the RL agent takes control. Moreover, it allows the player to gradually adapt to the task and discover relevant information about the game before the agent comes into play in order to to reduce the initialization errors of the agent and improve the stability of its learning. After these no-op actions they apply max-pooling over a 4 frames stacking (and then a 4 frames skip). They blank some parts of the images in order to not infer the reward with the scores and other information that are not representative of the dynamics of the games. Then, same as Mujoco, the details of parametrization are explained in the article.

3.3.2 The results

We are able to observe quite the same results on Atari games than on simulated robotics. The major change is the number of labels taken into account. Nevertheless, some differences can be seen on breakout, deep reinforcement learning does not perform well compared to real reward algorithm; but also in Qbert and it is especially the case for human labeling. Indeed, the short clips can be confusing and are not helpful to beat the first level. But on Enduro, human labeling outperforms every other experiments. If we look at it upside down, we can consider that the synthetic labeling and the Asynchronous Advantage Actor-Critic (A3C) fails to learn. It can be explained by the fact that passing cars is similar to random exploration which is easier for human to shape the reward. These poor performances, especially for human labeling, can be caused by human error, inconsistency between the human contractors labeling the same run, or an uneven rate of labeling by contractors. To

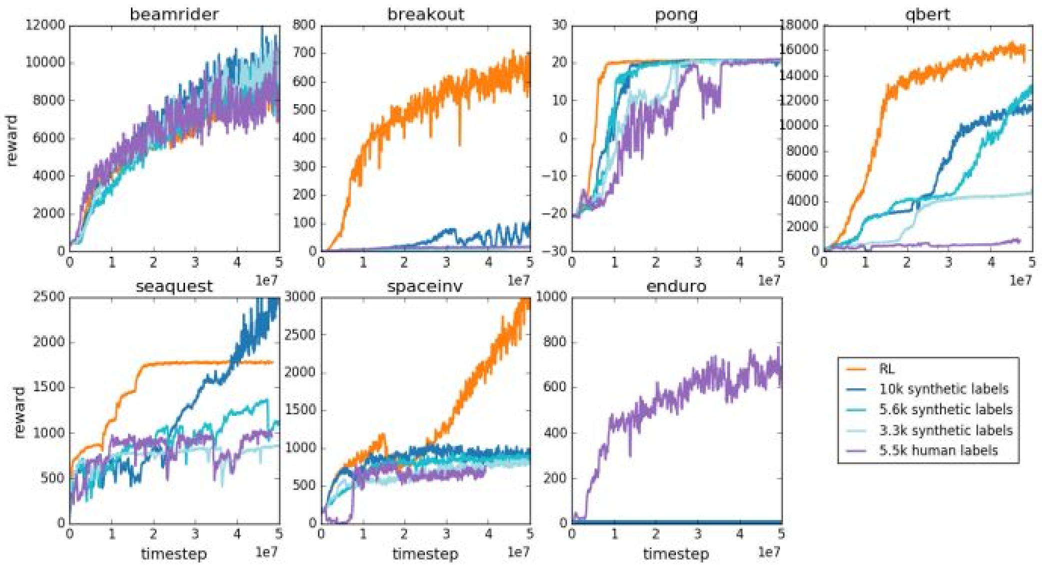


Figure 7: Results on Atari games

resolve the issue, using pipelines could be a solution.

3.4 Some Implementations

3.4.1 Famous implementations

The main implementation of this article is made by Matthew Rahtz, a research engineer at DeepMind here mrahtz. This implementation is really complete with a lot of possible tuning for comparisions. The paper selects video clips to show the user based on predicted reward uncertainty among an ensemble of reward predictors. Matthew Rahtz has shown experimentally that we had a higher chance of successful training by just selecting video clips randomly, thus he just chose segments randomly. One problem that we had with his code is the organisation of the code and the files, it make pretty hard to understand and modify in order to test on other environments and models.

An other awesome implementation is from an openia researcher Tom B Brown.

The specificity of this implementation is that if we choose to use human real feedback, we can choose between two videos on a webapp wich makes it really intuitive to use.

3.4.2 Our implementations

You can check our tests on the github : Implementation Git
We wanted to test the model on several environments and with different agents. To do so we based our implementations on the following repositories. depending on the parameters you take it might takes a long time to run (especially for A3C Cartpole)

- CartPole-V1 Implementation Pytorch which is an implementation were the user has to decide between two simulations which one is the best by entering either 1, 2 or 3 (if both simulation are similar). The human choice for the reward is taken into account from half of the total number of episodes. You can check the plots on the notebook, you can also run it with several parameters as we do. Finally, it seems that the humans preferences doesn't improves that much comparing to the model without human preferences, this environment might not be enough complex to really exploit the possibilities of the human helps.

Figure 8: CartPole Environment

From a high level point of view, A2C uses a single agent to interact with the environment and it updates the actor and critic networks simultaneously. The advantage of A2C is that it can achieve pretty good results with low computational resources. However it is not optimal and it can be slow for environments with high-dimensional state and action spaces.

Whereas, A3C uses multiple agents running in parallel to collect experiences and update the network parameters where each agent has its own copy of the network and interacts with the environment independently.The advantage of A3C is that it can scale to larger environments and achieve faster convergence. However, it is more computationally expensive than A2C.

- CartPole Implementation with A3C agent in this implementation we can compare A3C and A2C agents efficiency for the cartpole environment with tensorflow The model structure implemented for the actor critic agent is bellow 3.4.2

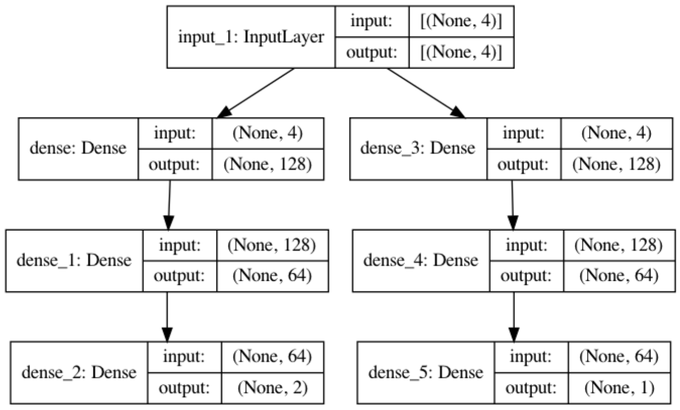


Figure 9: Actor Critic Agent Model

- Atari Implementation Pytorch for Atari implementation in pytorch (not ready for testing in our implementation)

- Atari Implementation Keras which performs Deep RL on Atari environment with an Asynchronous Actor-Critic Agent (A3C) with tensorflow.

3.5 Novel behaviors

The objective of the task is to see how effective the approach proposed by the authors is to solve tasks without the knowledge of any reward function. To see the effectiveness, the method was used on traditional Reinforcement Learning tasks with the same parameters as in the previous experiments. The results showed that the algorithm is able to learn novel complex behaviors. Indeed, they succeeded to make different robots from MuJoCo performing backflips or walking standing on one leg only with a restrict number of queries and amount of time required. For instance, the Hopper robot succeeded to make a backflip with 900 queries in less than an hour.

3.6 Ablation studies

This part is to understand the impact on performance of different parameters. To do so, six modifications are done to compare the algorithm's performance.

1. **Random queries:**
   The queries are picked uniformly at random instead of using preference elicitation mechanism (queries by disagreement

2. **No ensemble:**
   Only one predictor is trained instead of an ensemble of them. Therefore, the queries are randomly picked because the estimation of disagreement can not be done.

3. **No online queries:**
   The queries are trained at the beginning of the training phase. There is no more queries gathered through-out the process.

4. **No regularization:**
   No more $l2$ regularization. Dropout only.

5. **No segments:**
   The length of trajectory segments is 1. This is applied on robotics tasks only.

6. **Target:**
   No more fitting of the reward function $\hat{r}$ using comparisons. Using an oracle giving the true total reward on a trajectory segment, $\hat{r}$ is fitted on these rewards using mean squared error.

The figures presenting the performances showed one of particular interest, the poor performance of offline reward predictor training. A possible explanation is that the nonstationarity of the occupancy distribution leads the predictor to capture only part of the true reward. Thus, it maximizes this partial reward. Therefore, some strange behaviour can be observed and is undesirable. The main example is on Pong; the agent behaved to avoid losing points but did not intend to score some. This is one of the reason why human intervention during the process is beneficial.

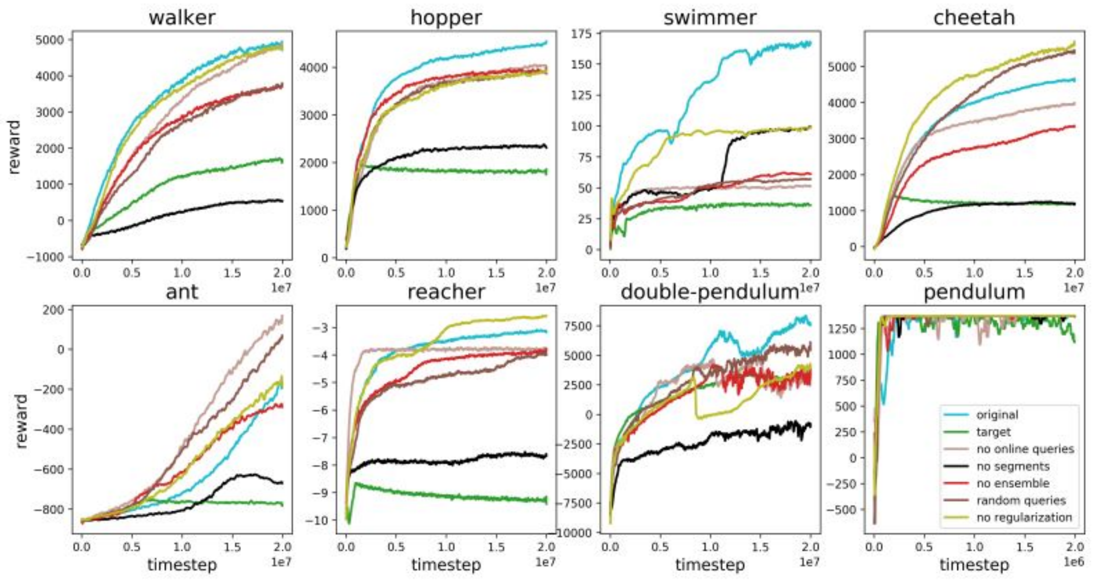


Figure 10: Performance results on MuJoCo robotics

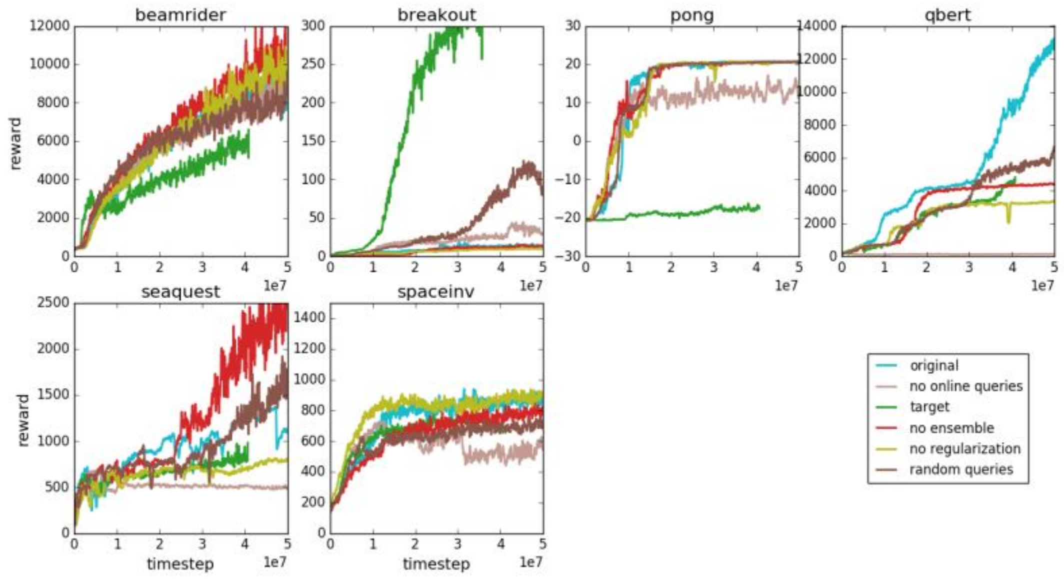


Figure 11: Performance results on Atari Games

It counters the static information given at the beginning, and becomes nonstationarity as the occupancy distribution is.

Besides, eliciting comparisons rather than absolute scores. The motive behind using comparisons is the ease for a human to provide consistent comparisons compared to provide a score or a metric. This is mostly the case for continuous control tasks or qualitative ones. Nevertheless, understanding how comparisons affect performance is important. What was observed is that neither comparison method nor target method outperformed each other.

Finally, to make comparisons video clips were used. Hence, this use of clips was modified to see the impact on performance. A first approach was using only frames. It was more difficult for humans to understand the context of the frames. As a result, the performance were better using video clips. However, the length of the clip is also impactful. Indeed, above a certain length, the human is able to understand and rate the clip on linear function basis. To overcome the issue of taking too much time getting the context and too much on rating the clip, the length chosen is the minimum time to have a linear evaluation time.

4 Possible extensions of the paper

In this article, the global dynamic of the policy relies on the human ("expert") providing explicit pairwise preferences between trajectories. The first idea to possibly extend this method could be to allow the expert to provide more complex feedback, we can imagine the human ranking multiple trajectories at once or providing feedback on individual state-action pairs in order to extract more information from the dynamic of the problem that the model is trying to learn.

Moreover, with the same approach in order to get more information on each trajectories, we could aggregate feed backs from multiple users instead of using only one feedback. But this extension needs more "experts" and we can loose the advantage that this method uses only "few" feed backs for great results.

Moreover, it can be really interesting to evaluate the performance of this method on more complex and subjective tasks. Indeed, in the article the authors are evaluating and comparing the quality of trajectory from objective tasks, for the Atari games it is pretty simple for all humans to say if a trajectory is worse than an other. We are curious about applying this approach to tasks where the quality of a trajectory is more difficult to evaluate, such as music composition which is a really subjective task.

Finally, the article [7], is an other paper about reward learning from human preferences. But here, their are combining two approaches to learn from human feedback: expert demonstrations and trajectory preferences. This is another extension of the article.

References

[1] Actor-Critic explanation. `https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f`.

[2] Advantage Actor-Critic and Asynchronous Advantage Actor-Critic explanation. `https://huggingface.co/blog/deep-rl-a2c`.

[3] Policy Gradient explanation. `https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146`.

[4] TRPO explanation. `https://towardsdatascience.com/trust-region-policy-optimization-trpo-explained-4b56bd206fc2`.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2017.

[7] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari, 2018.

[8] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[9] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.

[10] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.