A Survey on Deep Reinforcement Learning from Human Preferences

Virgile Rennard, Philippine Dolique

1

September 26, 2020

ABSTRACT

Reinforcement learning is the area of Machine learning where an agent interacts with an environment to maximize a reward signal. The advancement of Deep Learning made it so that more and more tasks previously impossible, notably for robotics, video games, and finance François-Lavet et al. (2018). However, some goals are too complicated to define as a reward function. Christiano et al. (2017) proposes a way to circumvolute this by indirectly crafting the reward function by the way of human preferences between two short video segments. This is done in such a way that the human interaction can stays minimal, whithout needing experts input, teaching novel complex behavior with less than an hour of human time, and can be applied to state of the art Reinforcement learning systems.

Key words. Reinforcement Learning – Deep Learning – Human preferences – Atari – MuJoCo Simulated Robotics

1. Introduction

Reinforcement learning is a specific area of Machine learning which is taught with neither supervised or unsupervised methods, but rather learning while it is training. There has recently been a lot of success in scaling RL to large problems; whether it be boardgames, Silver et al. (2017), Schrittwieser et al. (2019) , videogames, Vinyals et al. (2019), Lample and Chaplot (2016), or in health, Yu et al. (2019). Moreover, the recent upsurge in deep learning was applied to RL as well, bringing about the Deep Reinforcement Learning François-Lavet et al. (2018) which made leaps and bounds for the Reinforcement Learning community.

However, as was explained, the framework of reinforcement learning needs an explicit reward function for the agent to learn how to interact with the environment. An explicit reward function is therefore needed to use most of the usual optimisation methods Sutton and Barto (1998). Albeit, some behavior are too complex to model with a reward function. As an exemple, if you want to train a robot to do a backflip, or to cook Paella, the way to construct an acceptable reward function isn't defined, as it needs to be a function of the robot's sensors. One way to do so is to build a simple reward function that captures the intended behavior, but this will likely not satisfy our preferences and lead to other problems down the road; Russell (2016).

In 2000, Ng and Russell (2000) explained how to get a reward function from a set behavior by using "inverse reinforcement learning", which, given a measurement of an agent's behavior over time in different circumstances, measurement of the sensory inputs to that agent and a model of the environment models the reward function being optimized. This is useful for the use of RL in computational models for animal and human learning, supported by behavioral studies and neurophysiological evidence that reinforcement learning occurs in bee foraging Montague et al. (1995), or to construct an intelligent agent that can behave successfully in particular domains by using the behavior of experts in this domain. The resulting reward function can be used to train the agent with Reinforcement Learning. However, this approach is not applicable to behaviors that are too difficult for humans to demonstrate.

In Deep Reinforcement Learning from Human Preferences Christiano et al. (2017) proposes another way to approach Ng and Russel's inverse RL idea for such tasks. The idea being that, while human may not be able to demonstrate some complex behaviors such as a robot having to cook paella with many degrees of freedom and non human morphology, they are able to recognize from video segments whether the robot is doing well on a given task or not, by providing feedback on the system's current behavior and using it to define the task. This is an usual idea in reinforcement learning, that has however always been impracticable, as it needs an excessive amount of human time in order to get enough feedback to have a good enough reward function. This is one of the main pitfall that Christiano et al. (2017) addresses.

The idea is to create an algorithm that fits a reward function to the human's preferences while simultaneously training a policy in order to optimize the current predicted reward function. The Human controller will watch short video clips of the agent's behavior and choose which one is preferable to the other. Comparisons are easier for humans to do in some domains and still gives equally useful data for learning. Moreover, comaring short clips is nearly as fast as annotating one.

Fig. 1. The reward predictor is trained asynchronousy from comparisons of trajectory segments, and the agent maximizes the predicted reward

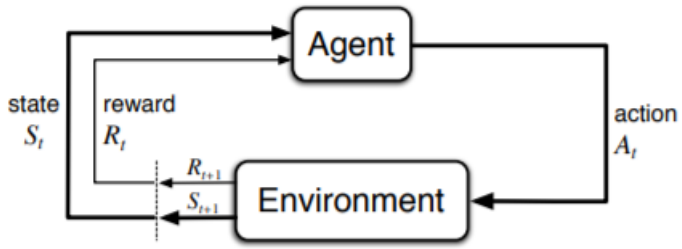

Fig. 2. Usual flow of a RL Algorithm for comparison (by Sutton and Barto (1998))

It is important to note that there is a lot of previous work on reinforcement learning from human ratings, such as Akrour et al. (2011), Akrour et al. (2012) to cite a few. In Christiano et al. (2017), the algorithm is based on a modified approach to Schoenauer et al. (2014), where they present Programming by feedback, which involves a sequence of interactions between the active computer and the user, used on gridworld, and on the Nao robot, a robot with four degrees of freedom and small discrete domains. This is done on entire trajectories instead of small segment of videos, and asks a lot of human time. This paper reduces needed human time and is suitable for robots with dozens of degrees of freedom.

The main result of this paper is a way to scale human feedback to deep reinforcement learning method, to learn much more complex behaviors.

2. The Method

The basic approach is to learn a reward function from human feedback and then to optimize that reward function. This is a solution to sequential decision problems without a specified reward function that can

- enable one to solve task where the desired behavior can be recognized without having to demonstrate it
- allow agents to be taught by non-expert users
- scale to large problem
- doesn't require too much feedback from the users

The agent interacts with the environment over a sequence of steps t, receiving an observation $o_t \in \mathcal{O}$ from the environment, and sending an action $a_t \in \mathcal{A}$ to the environment accordingly.

As is shown in fig.2, in reinforcement learning is a reward signal expected from the environment. This reward $r_t \in \mathbb{R}$ being the central value the agent tries to maximize; specifically the agent would try to maximize the discounted sum of these rewards, as their values decays through time.

This assumption is removed in the case of human interaction. As is illustrated in fig.1, it is not assumed that the environment produces a reward signal, and instead assumed that the human overseer can give a preference between trajectory segments.

Definition 1 *Let $\mathcal{O}$ be the set of observations and $\mathcal{A}$ be the set of actions. A trajectory segment is a sequence of observations and actions σ where :*

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

The goal would be for the agent to produce trajectories that are preferred by the human, all the while making the fewest possible queries so as to save time for the human operator. In order to evaluate the algorithms behavior is an ordering relationship needed between the segments.

Definition 2 *We define an order relationship $\succ$ between trajectories. Preferences $\succ$ are generated by a reward function $r : \mathcal{O} \times \mathcal{A} \longrightarrow \mathbb{R}$. We say that $\sigma^1 \succ \sigma^2$ if :*

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

that is when :

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

This order relationship gives an "easy" way to quantitatively evaluate the algorithms behavior, as the human's preferences are generated by the reward function r that the agent tries to maximize. It follows that if we know the reward function r, the agent can be evaluated quantitatively, reaching a reward as high as if we used traditional reinforcement learning methods to optimize it.

This quantitative way of evaluating has its flaws however, as the reward function isn't necessarily available. Note that this is when the approach in Christiano et al. (2017) is useful. When it happens, the only way to evaluate segments σ is qualitatively with the human eye. It can start from a goal expressed via text that a human has to evaluate the agent's behavior against with videos of the agent attempting to fulfill that goal.

Schölkopf et al. (2007) presents a trajectory segment comparison model that will form a base for the Deep Learning RL from human perspective to build on, as it will be tweaking it by assuming that the system can be reset in arbitrary states, in order to have different starting states.

For each point in time, the method maintains a policy $\pi : \mathcal{O} \longrightarrow \mathcal{A}$ and an **estimation** of a reward function $\hat{r} : \mathcal{O} \times \mathcal{A} \longrightarrow \mathbb{R}$ tuned by neural networks which will do the following updates :

- (1) The policy π interacts with the environment to produce a set of trajectories $\{\tau^1, \ldots, \tau\}$. π is updated by traditional RL algorithms in order to maximize the sum of predicted rewards $r_t = \hat{r}(o_t, a_t)$
- (2) We select pairs of segments (σ^1, σ^2) from the trajectories $\{\tau^1, \ldots, \tau^i\}$ produced in the first step and send them to a human for comparison
- (3) The parameters of the mapping $\hat{r}$ are optimized via supervised learning to fit the comparisons collected from the human so far

The processes run asynchronously, where the trajectories flow from (1) to (2) to (3) to (1)

These three steps fully define the algorithm, and it is important to go more in depth in explaining them in order to both understand them and implement the methods. The three following parts will be explaining each steps.

2.1. The Optimization step

Once the estimated reward function $\hat{r}$ has been used to compute the rewards, the only thing left to do is use traditional reinforcement learning. In choosing the algorithm, the fact that the reward function $\hat{r}$ is non-stationary has to be taken into account, leading one to taking a method which is robust to changes in the reward function. This is why policy gradient methods are used.

Specifically, for the two main tasks; Atari Games, and Simulated robotic tasks, are used respectively Mnih et al. (2016) **advantage actor-critic**, and Schulman et al. (2015) **trust region policy optimization**. In each case, the parameter settings which have been found to work well for traditional reinforcement learning tasks are used. In the case of the TRPO, the entropy bonus is however adjusted on a case by case basis, as it can empede proper exploration, leading to inadequate exploration for a changing reward function. The rewards produced by $\hat{r}$ are normalized to have zero mean and a constant standard deviation.

2.1.1. A closer look into Trust Region Policy Optimization (TRPO)

As one of the main result of Christiano et al. (2017) is obtained by following the algorithm developed in Schulman et al. (2015), taking a more in depth look at how the algorithm functions seems fundamental to our understanding.

TRPO is effective for optimizing large nonlinear policies such as Neural Networks, and usually gives monotonic improvement with small hyperparameters tuning.

TRPO basically comes from the following idea : You can minimize a surrogate loss function in order to guarantee policy improvement with a certain step size. Once this is done, a series of approximations to the theoretically-justified algorithm is done, leading to a practical algorithm (the TRPO). We will briefly discuss the single-path method in order to get a sense of how the algorithm functions. This is specifically in the model free setting. The other variant being the vine method, which would require the system to be restored to particular states.

With an infinite horizon discounted MDP, defined by $(S, \mathcal{A}, P, c, \rho_0, \gamma)$, with S and A as before, and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ the transition probability distribution, $c : \mathcal{S} \longrightarrow \mathbb{R}$ the cost function, $\rho_0 : \mathcal{S} \longrightarrow \mathbb{R}$ the distribution of the initial state s_0 (the probability of the system starting in a state), and $\gamma \in (0, 1)$ the discount factor. Let moreover $\pi : \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$, be a stochastic policy, and $\eta(\pi)$ be its expected discounted cost. Let us also denote $Q_\pi(s_t, a_t)$ the classical action value function, $V_\pi(s_t)$ the value function, and $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$ be the advantage function.

The expected cost of another policy $\tilde{\pi}$ in terms of the advantage over π is given by $\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$ with $\rho_\pi(s) = (P(s_0 = s) + \gamma(P(s_1 = s) + \dots)$. This implies that any policy update $\pi \longrightarrow \tilde{\pi}$ that doesn't have a positive expected advantage at every state s is guaranteed to either reduce η or leave it constant. This implies the result that the update performed by exact policy iteration improves the policy if there is at least one state action pair with a negative advantage value and nonzero state visitation probability. The dependency of $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes it difficult to optimize in the approximate setting, leading to optimizing the local approximation to η : $L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$.

L_π uses ρ_π and not $\rho_{\tilde{\pi}}$, ignoring changes in state visitation density due to changes in the policy. If the policy is parametrized π_θ and $\pi_\theta(a|s)$ is a differentiable function of the parameter vector θ, then L_π is equal to η to the first order. This implies that a small step $\pi_{\theta_0} \longrightarrow \tilde{\pi}$ that improves the old L_{π_θ} will improve η. The problem being that we do not know how large the step should be.

With π_{old} the current policy, and $\pi' = argmin_{\pi'} L_{\pi_{old}}(\pi')$, the new policy is given by $\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha\pi'(a|s)$. The following bounds holds :

$$\eta(\pi_{new}) \leq L_{pi_{old}}(\pi_{new}) + \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{1}$$

With ϵ the maximum advantage of π' relative to π

All this results are proved in Kakade and Langford (2002)

This implies that a policy update that improves the right hand side is guaranteed to improve the true expected cost objective η. This can be extended to general stochastic policies by replacing α with a distance measure between the policies. This distance is the total variation divergence, defined by $D_{TV}(p\|q) = \frac{1}{2}\sum_i |p_i - q_i|$ for discrete probability distributions p and q. The main result of the TRPO paper is that the bound holds when we set $\alpha = D_{TV}^{max}(\pi_{old}, \pi_{new}) = max_s D_{TV}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s))$. The square of the total variation divergence is always lower than the Kullback Leibler divergence, which gives us this bound

$$\eta(\tilde{\pi}) \leq L_\pi(\tilde{\pi}) + C D_{KL}^{max}(\pi, \tilde{\pi}) \text{ where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$$

Which gives us the following approximate policy iteration scheme algorithm.

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \dots$ until convergence **do**
Compute all advantage values $A_{\pi_i}(s, a)$.
Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_\pi \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi)\right]$$
$$\text{where } C = 4\epsilon\gamma/(1-\gamma)^2$$
$$\text{and } L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$$

end for

The TRPO algorithm is an approximation to the Algorithm 1 which uses a constraint on the KL divergence rather than a penalty to allow large updates in a robust method.

In the TRPO, the problem that is solved is :

$$min_\theta \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s, a)$$
$$\text{such that } D_{KL}^\rho(\theta_{old}, \theta) = \mathbb{E}[D_{KL}(\pi_{\theta_{old}(\cdot|s)}\|\pi_\theta(\cdot|s))] \leq \delta$$

which is equivalent to solving

$$min_\theta \mathbb{E}_{s\sim\rho\theta_{old}, a\sim q}[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{old}}(s, a)]$$
$$\text{such that } \mathbb{E}_{s\sim\theta\rho\theta_{old}}[D_{KL}(\pi_{\theta old}(\cdot|s)\|\pi_\theta(\cdot|s))] \leq \delta$$

the single path sampling scheme will replace the expectation by sample averages and the Q by an empirical estimate. To do so, one must collect a sequence of states by sampling

$s_0 \sim \rho_0$ and simulate the policy $\pi_{\theta old}$ for some number of timestep to generate a trajectory $s_0, a_0, \ldots, a_{T-1}, s_T$. This way, $q(a|s) = \pi_{\theta old}(a|s)Q_{\theta_{old}}(s,a)$ is computed at each state action pair by taking the discounted sum of future cost along the trajectory. Another option is the vine sampling, which is described in Schulman et al. (2015), which is more effective but more time consuming.

The resulting algorithm is the following :

- Use the single path procedure to collect a set of state action pairs along with monte carlo estimates of their Q values
- Average over samples, and construct the estimated objective and constraint in our optimization problem
- Approximately solve this constraiend optimization problem tu update the policy's parameter vector θ by using a conjugate gradient algorithm with a line search.

In Christiano et al. (2017), the algorithm is used with $\gamma = 0.995$, $\lambda = 0.97$, and the reward predictor is a two layer neural network.

2.1.2. A closer look into advantage actor critic methods

As with TRPO, we are going to take a more in depth look at how the ideas presented in Mnih et al. (2016) relate to the playing of Atari Games.

In their paper is proposed a lightweight framework for deep RL that uses asynchronous gradient descent for optimization of deep NN controllers. Specifically the asynchronous variant of actor critic used in Christiano et al. (2017).

For Deep RL, most algorithm store the agent data in an experience replay memory. It is however costly in memory and computation, and requires off policy learning algorithm. The asynchronous actor critic method does not stand on experience replay, but rather, executes multiple agents simultaneously on multiple instances of the environment. This is all done parallelly, decorrelating the agent's data into a more stationary process. This way, on policy algorithm such as actor critic methods can be applied robustly do deep NN. This algorithms can be ran on a single machine with a standard multi-core CPU.

We remind that the action value function is $Q_\pi(s,a) = \mathbb{E}[R_t|s_t = s, a]$, with the optimal value function $Q^*(s,a) = max_\pi Q_\pi(s,a)$ giving the maximum action value for a state s and an action a for any policy.

The action value function is represented by an approximation $Q(s,a;\theta)$, where the updates to θ can be gotten from multiple RL algorithms, such as Q-Learning. Q-learning is a common RL method where the goal is to approximate Q^*, where the parameters are learned by minimizing a sequence of loss functions $L_i(\theta_i) = \mathbb{E}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i)]^2$. This is specifically a one step method, as it only takes into account a one step return. This is slower that n-steps returns.

$b_t(s_t) \approx V^\pi(s_t)$ is the baseline used to lower the variance estimate of the policy gradient. Indeed, using $R_t - b_t$ as an estimate of the advantage function of action a_t in state s_t $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ as R_t is an estimation of the action value function, and b_t an estimation of the value function.This is an actor critic architecture where the policy π is the actor and the baseline b_t is the critic.

Specifically, the algorithm used in Christiano et al. (2017) is what Mnih et al. (2016) calls the A3C. It maintains a policy $\pi(a_t|s_t;\theta)$ as well as an estimate of the value function $V(s_t;\theta_v)$. It operates in the forward view and uses a mix of n-step returns to update both the policy and the value-function. They are updated after t_{max} actions of when a terminal state is reached. The update is $\nabla_{\theta'} ln(\pi(a_t|s_t;\theta'))A(s_t,a_t;\theta,\theta_v)$, with the estimate of the advantage function being $\sum_i^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v)(V(s_t;\theta_v))$, where k is upper bounded by t_{max}. This algorithms uses parallel actor-learners and accumulated updates in order to improve the training stability.

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t+1$
    $T \leftarrow T+1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

In Christiano et al. (2017), this algorithm is used in its synchronous form, with an entropy bonus $\beta = 0.01$, a learning rate of 0.0007 decayed linearly to be 0 in 80 million timesteps, $n = 5$ steps per update, and $N = 16$ parallel workers. The discount rate was $\gamma = 0.99$, and the policy gradient used Adam.

2.2. The second step : expliciting the preferences

The Human overseer is sent many visualisation pairs of two trajectory segments in the form of short movie clips, that are each from one to two seconds long.

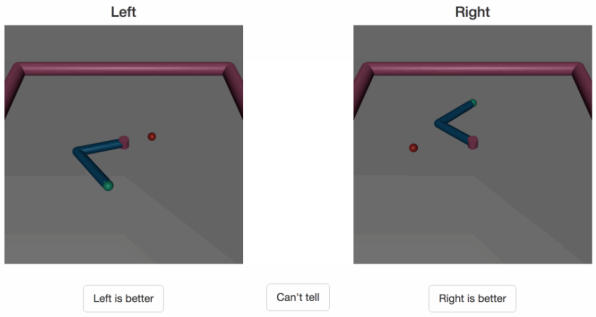

Fig. 3. A comparison of two short clips. Here the robot tries to reach a ball as fast as possible.

At this point, the overseer simply indicates whether a segment is better than another, equally good, or incomparable.

This opinion is stored in a database $\mathcal{D}$ of triplets $(\sigma^1, \sigma^2, \mu)$, where σ^1 and σ^2 are the two compared segments, and μ is a "probability" distribution over the set $\{1, 2\}$, which indicates the probability of each segment being preferred. If the overseer chose one segment as preferable, the entire mass of μ is on the corresponding value; if they are equally preferable, $\mu \sim \mathcal{U}_{[|1,2|]}$. If it was considered incomparable, the comparison is not included in $\mathcal{D}$.

2.3. The third part : Fitting the reward function

The estimation of the reward function $\hat{r}$ can be interpreted as a predictor of preferences. Indeed, $\hat{r}$ can be seen as a latent factor explaining the human's judgement, and assume that the human's probability of choosing a segment σ^i depends exponentially on the value of the latent reward summed over the length of the clip.

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

Which is the softmax function. This is without the use of discounting that we can usually see in RL, which translates the fact that the human's preference is independent to when the clip plays.

$\hat{r}$ is taken so as to minimize the cross entropy loss between these predictions and the real human labelling.

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

This is similar as equating rewards with a preference ranking scale, in the same idea as the chess ELO ranking Elo (1978). Indeed, as the differences in points on the ELO scale in chess represents the probability of each player winning, so does the difference in predicted reward of two segments estimate the probability of one segment being chosen over the other by a human.

As is usual in machine learning, the algorithm fits an ensemble of multiple predictors trained on $|\mathcal{D}|$ triplets sampled from $\mathcal{D}$, in order to have an estimate for the reward function when the predictors are normalized and averaged.

One of the problems with the use of the softmax function is that it doesn't take into account the probability of human error. One way to take it into account is to add stochasticity, by adding a chance that human responds randomly uniformly.

Queries are based on an approximation to the uncertainty in the reward function estimator. With many pairs of trajectories, each reward predictors is used to predict which segment will be preferred for each pairs. The trajectories for which the predictions have the highest variance will be selected.

2.3.1. The selection of queries

The training is done with ensemble learning. This means that multiple predictors are stacked on each comparisons in order to reduce the overall variance and end up with the best model possible. It is important to note that, while it is worthwile most of the time, it can sometimes be detrimental to the final output.

The selection of queries is simple. Intuitively, the queries that have the higher variance over the different predictors are the one where the most interesting behaviors shows. They should be worked on before the others. Therefore, k trajectories (σ_1, σ_2) are selected, and the training works first on the ones where the predictor shows the highest variance between predictions.

3. Quick overview of experimental result

As an attempt to be exhaustive, we will now give a brief overview of the experimental results obtained in the original paper Christiano et al. (2017).

The experiments are done by giving overseers about two sentences of description of a task before having to compare hundreds to thousands of pairs of trajectory segments. **this takes only between 30 minutes to 5 hours**.

Along with the actual human experience is the experiment ran with queries sent to synthetic oracle whose preferences reflect the exact reward of the task. This means that instead of sending the comparison to an actual human, it is instantly answered with the better segment for the task. This will of course all be compared to a baseline of Reinforcement learning using a real reward.

It is important to note that **in the case where an exact reward function is available, the goal of the algorithm is to do as well as the RL algorithm**. Indeed, once again, the strength of this approach comes in for tasks where the exact reward function isn't available. This is done for comparison purposes. **With this method, having results similar to the ones with the reward function available is great, as the optimization is done with no information on it**.

In the context of Sutton and Barto (1998), the experiments are not done as an episodic task; Indeed, having termination condition in for an example a video game would surely give information about the task, this is not something that is wanted for the experiment. As such, other signals such as the score in a video game is hidden, as it would give away the entire reward function. This way, the only available information is the human's preferences.

3.1. Significant results

When it comes to the experiments done with simulated robotics, this is done by maximizing the reward function with classic RL methods, with 700 queries sent to humans, and 350,700,1400 queries sent to an oracle. For most of the cases, the 1400 synthetic queries work better than anything else, resulting in this kind of output. Specifically, here are the results obtained on the MuJoCo (Todorov et al. (2012)) walker task : Here, the center of the robot is the joint where the three limbs meet. It has to move to the right, while being as high as possible.

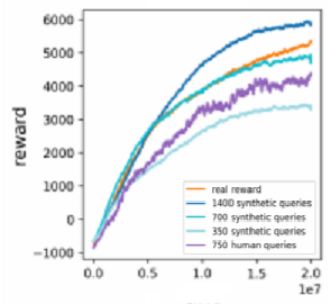

Fig. 4. Results on the MuJoCo simulated robotics walker task measured on the tasks reward.

However, for some specific complex tasks, even with the actual reward function given and the oracle giving synthetic responses, the human feedback can outperform all other methods. Specifically, the experiments output on the Ant task is maximized by the human queries. The Ant task is one where the robot has to be standing upright and moving to the right.

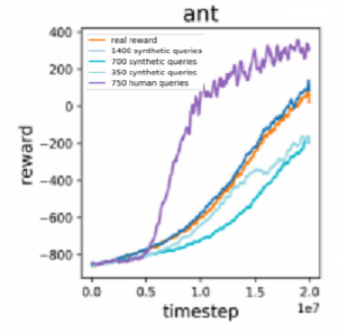

Fig. 5. Results on the MuJoCo simulated robotics ant task measured on the tasks reward.

The results for the Atari video game are similar.

However, all those results are for behaviors that were already known and trained on, with the reward function known. Some experiments were done to implement novel behaviors. While comparison with traditional RL tasks with known rewards helps us understand if the method is effective, the goal of this method is to train on tasks with no available reward function. In the paper, the MuJoCo robot has succesfully been trained to do successive backflips, while the Cheetah robot was able to learn how to move forward on one leg. Both of this have been learned with less than an hour of human supervision.

4. Our experimentation

In the scope of this work, we have made our own implementation of deep reinforcement learning with human preferences. The model was trained in pytorch. It is available on github https://bit.ly/2wY7cXz Alg[1]. With this model, we have trained on the simple cartpole task with the gym openai environment. MuJoCo and Atari tasks works in the same way, but the training is too expensive to run on light computers.

The cartpole task is a simple task where a pole stands on a moving cart, and has to avoid falling. The more upright the pole, the higher the reward. This is a very simple task and is often used as a benchmark for classical Reinforcement Learning methods.

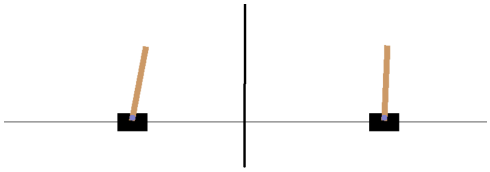

Fig. 6. An illustration of the cartpole task and our implementation. Two windows show short segments of the cartpole task, the human superviser has to pick the best one. An animated illustration as well as the code to train it is available on the github.

In order to compare, we trained this task with both the human supervisor and traditional RL. The results are presented below.

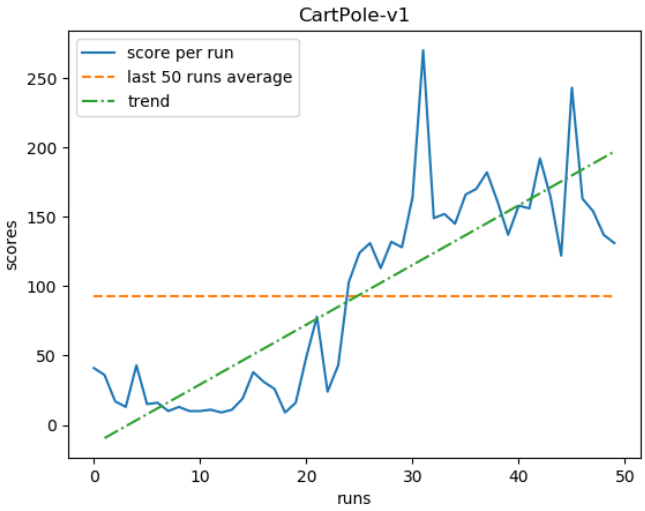

Fig. 7. The cartpole task, trained with traditional reinforcement learning methods

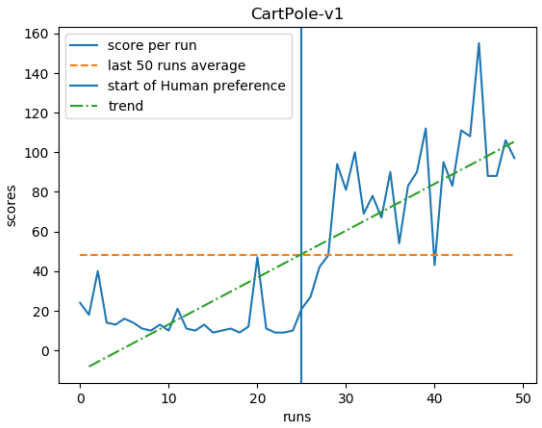

Fig. 8. The cartpole task, trained with human supervision and our Alg[1]

In the case of the cartpole, we can see that the traditional RL methods outmatch our implementation. This is to be expected as, as was explained thoroughly, when the explicit reward is known, the goal is not to beat, but match the traditional methods. Moreover, the cartpole exercise is one with a very simple reward compared to most MuJoCo task, and is therefore learned quickly by classical RL. However, the Human Supervision did lead, in only 15 minutes, to satisfying results, as the pole stood upright and did not move too much.

As was said, going further to compare on more complex tasks would require more powerful computers. The results were however still satisfactory as the goal was met without knowing the underlying reward function.

5. Conclusion

We have presented that there is a way to scale up preference elicitation in Reinforcement learning to state of the art level. This is a radical step for Reinforcement Learning to be applied to tasks that are, in the real world, too complex to learn with traditional RL methods. One way to go from this point is to find how to improve the efficacity of this learning, in order for it to be applied to all kind of tasks. One pitfall being that for complex tasks, expert human supervision would be required. Being able to make it so that learning from human preferences is as efficient as from

traditional RL methods would launch the applications of RL to new heights, as it would finally be usable on complex tasks.

References

Our github implementation. https://github.com/PhiDole/DeepReinforcementLearningFromHumanPreferences?fbclid=IwAR0gIAF4kc1SML13csETJGkXj45xuxfNr_f_ykgsfJrL5W2NOv7DH-OMfp4.

Riad Akrour, Marc Schoenauer, and Michele Sebag. Preference-based policy learning. In *Proceedings of the 2011th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECMLPKDD'11, pages 12–27, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23779-9. . URL https://doi.org/10.1007/978-3-642-23780-5_11.

Riad Akrour, Marc Schoenauer, and Michèle Sebag. APRIL: active preference-learning based reinforcement learning. *CoRR*, abs/1208.0984, 2012. URL http://arxiv.org/abs/1208.0984.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4299–4307. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7017-deep-reinforcement-learning-from-human-preferences.pdf.

Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. Arco Pub., New York, 1978. ISBN 0668047216 9780668047210. URL http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018. URL http://arxiv.org/abs/1811.12560.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 267–274, 2002.

Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016. URL http://arxiv.org/abs/1609.05521.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/mniha16.html.

Pat Montague, Peter Dayan, Christophe Person, and Terrence Sejnowski. Bee foraging in uncertain environments using predictive hebbian learning. *Nature*, 377:725–8, 11 1995. .

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL http://dl.acm.org/citation.cfm?id=645529.657801.

Stuart H. Russell. Should we fear supersmart robots? *Scientific American*, 314 6:58–59, 2016.

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. URL http://proceedings.mlr.press/v32/schoenauer14.html.

Bernhard Schölkopf, John Platt, and Thomas Hofmann. *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference (Bradford Books)*. The MIT Press, 2007. ISBN 0262195682.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model, 2019.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/schulman15.html.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

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

Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019. .

Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement learning in healthcare: A survey, 2019.