Reinforcement Learning : Deep Reinforcement Learning from Human Preferences

Article written by : Paul F Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, Dario Amodei

Master DS group names : Benjamin Cohen, Raphaël Gervillié

18th of February, 2022

Course : **MAP670C - Reinforcement Learning (2021-2022)**

Abstract:

Interacting with the real world may only be thought off if we understand how to communicate the underlying complex purposes of each systems and how they interact with each other. Reinforcement learning based systems interact with their environments and thus we explore in this article goals of human preferences between several trajectory segments. Indeed we will see that this method can effectively solve complex RL tasks without access to the reward function, including Atari games and simulated robot locomotion, while providing feedback on less than 1% of our agent's interactions with the environment. This reduces the cost of human monitoring enough that it can be practically applied to modern RL systems. To demonstrate the versatility of the approach, we show that we can successfully train new complex behaviors with about an hour of human time. These behaviors and environments are significantly more complex than any previously learned behavior and environment from human feedback.

Introduction

- The last decades has seen huge advances in scaling reinforcement learning (**RL**) to major problems in domains where the **reward function** is known and specified. Unfortunately, many tasks include complex, unclear, or difficult-to-define objectives. Overcoming this limitation may significantly increase the potential effect of **Deep RL** and expand the range of machine learning even wider.
- If we want to use reinforcement learning to **train a robot** to clear a table or shuffle eggs. It is not yet clear how to build a proper reward function for the robot sensor. We can design a simple reward function that is close to the behavior we want, but this often results in behavior that optimizes the reward function without actually satisfying what we expect the robot to do. This is the fundamental difficulty raising concerns about **discrepancies between RL system values and the actual goals that we want to achieve**.
- If we could successfully **communicate our true goals to the agents**, it would be an important step in addressing these issues. IRL is about learning from humans it represents one way to achieve learning the agents' objectives, values, or rewards by observing its behavior. This is what we call **inverse reinforcement learning** seen as a method to extract the reward function [7] [9]. The purpose is to learn the decision-making process to generate behaviors that maximize the predefined reward function. Basically, the goal is to extract the reward function from the observed behavior of the agent.
- **This reward function can then be used to train an agent with RL**. More directly, we can use **imitation learning** to replicate its proven behavior. However, these approaches cannot be directly applied to behaviors that are difficult for humans to prove.(such as controlling a robot with many degrees of freedom but a very nonhuman form).
- Another approach is to allow humans to provide **feedback on the current behavior of the system** and **use that feedback to define the task**. This fits into the reinforcement learning paradigm, but using human feedback directly as a reward function is exorbitantly expensive for **RL**. Such a system would require hundreds or thousands of hours of experience. To use human feedback to train our RL systems we need to reduce the amount of feedback required from multiple orders of magnitude. Our approach is to **learn the reward function from human feedback and optimize its reward function**. This basic approach has been considered previously, but we confront the challenges involved in scaling it up to **modern deep RL** and demonstrate by far the most complex behaviors yet learned from human feedback. In summary, we need a solution to the sequential decision problem without a clear reward that is able to :
  - Solve tasks that can only **recognize the desired behavior** but where we do **not necessarily need to prove it.**
  - Allow agents to be taught by **inexperienced users**.
  - **Scale** to big problems.
  - **Not being too greedy on users' feed backs**

- The algorithm uses **human's preferences to fit the reward function while training a policy to maximize the current predicted reward function**. Our algorithm fits a reward function to the human's preferences while simultaneously training a policy to optimize the current predicted reward function (see Figure 1).

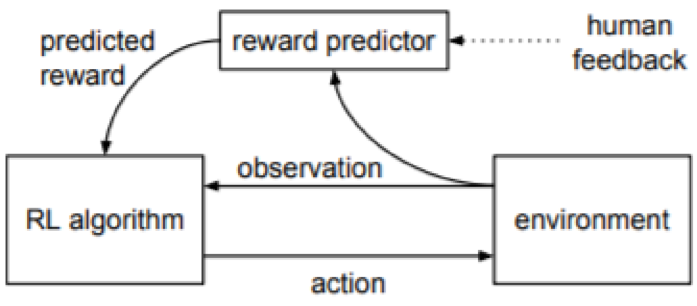


Figure 1: **Reward predictor trained asynchronously on comparisons of trajectory segments.**

- Then **we ask humans to compare short video clips of the agent's behavior**, rather than using an absolute numerical score. Using human ratings of how much the robot does what we expect him to do makes it easier to make comparisons between the trajectories while being equally useful for learning our human inner preferences. **Comparing different short videos is nearly as fast as comparing single states and they even show that the resulting comparisons are utterly more helpful**.

- Collecting feedback online helps **increasing the system's performance** and **avoids** exploiting weaknesses of the learned reward function implying to avoid weird behaviors that would fulfill the task in a not human way. The experiments leaded in the article are revolving around two subjects : **Atari games in the Arcade Learning Environment** [3], and **MuJoCo : A physics engine for model-based control** [11].

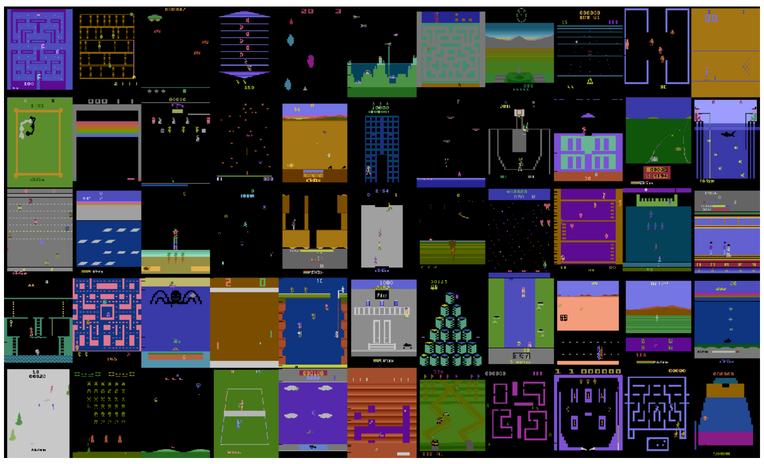


Figure 2: **55 games of the Atari Arcade Learning Environment**

- The article shows that we need only need a **small scale of feedback from inexperienced users**, needing from fifteen minutes to five hours to learn most of the original RL tasks **even when the reward function is not well specified**. Then they consider new behaviors in each domain, such as performing a backflip or driving with the flow of traffic. They show that **their algorithm can learn these behaviors in about an hour of feedback** even though it is unclear how to conceive a reward function for each task that would encourage the good behaviors.

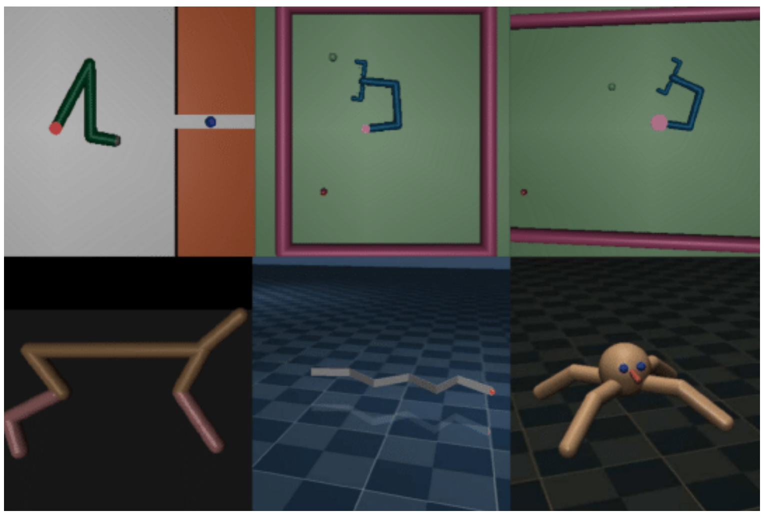


Figure 3: **Robotic manipulation and locomotion environments in MuJoCo simulator**

‘

- Many studies of **RL studying human ratings or rankings**, including [2] consider the same general problem of reinforcement learning as seen in the article focusing on **human preferences** rather than **absolute reward values**, and optimizing using human preferences in settings other than reinforcement learning. Their algorithm follows the same basic approach as in [1]. considering **a robot with four degree of freedom with continuous domains and assumes that the reward is linear in the expectations of hand-coded features**.

- In the article, they even consider physics **tasks with dozens of degrees of freedom** and Atari tasks with no hand-engineered features.

  As **their environments are complex**, they need to use various RL algorithms and reward models to cope with **several algorithmic tradeoffs**. In [2] they obtain preferences over the whole trajectories rather than short clips. Nevertheless in the article they gather **about two orders of magnitude more comparisons**, their experiments require less than one order of magnitude more human time.
  One of the other difference that appears between their works is their **focus on changing their training procedure to cope with the nonlinear reward models** and modern deep RL using **asynchronous training and ensembling**.

- They **fit** this reward function **using Bayesian inference** using **synthetic human feedback** which was drawn from their **Bayesian generative model** while they performed experiments with **feedback drawn from non-expert users**.
  This method can be extended to even more complex tasks for example when we can work with real human experiments involving reinforcement learning from actual human feedback, although **the training only occurs when the human trainer provides feedback** and games like **Atari** games require thousands of hours of experience to learn a high-quality policy.
  They also **need to learn a reward function**, however they consider simpler settings where the desired policy can be learned somewhat quicker. Their work could also be seen as a specific instance of the cooperative **inverse reinforcement learning**. This framework considers a **two-player game between reward function**. In their setting

the human is only allowed to interact with the game by stating their comparisons to all prior work, their key contribution is to scale human feedback up to deep reinforcement learning and to learn more complex behaviors.

Assumptions and Method

Situation and purpose

- We consider that we are constantly interacting with an environment over **different time steps** ; **at each time t** the agent receives an **observation** $o_t \in O$ from the environment and then sends an **action** $a_t \in A$ to the environment.

  Usually the environment also supplies a **reward** $r_t \in R$ and the agent's goal is to **maximize the discounted sum of rewards**. Instead of assuming that the environment produces a reward signal, they assume **a human oversees** the experiment and can tell a human way of completing the task between **several trajectory segments**.

  A **trajectory segment** is a sequence of **observations** and **actions**, $\sigma = ((o_0, a_0), ... ,(o_{k-1}, a_{k-1})) \in (O \times A)^k$.

  We write $\sigma^1 \succ \sigma^2$ to denote the fact that we prefer trajectory segment σ^1 to trajectory segment σ^2. The purpose of the agent becomes to find the best trajectories induced by a human choosing, and make as less queries as possible to the humans answering. They evaluate their algorithms' behavior in two ways:

  Quantitative: We say that preferences $\succ$ are generated by a reward function : $r : O \times A \rightarrow R$ if $((o_0^1, a_0^1), ... ,(o_{k-1}^1, a_{k-1}^1)) \succ ((o_0^2, a_0^2), ... ,(o_{k-1}^2, a_{k-1}^2))$ whenever $r(o_0^1, a_0^1), ... ,r(o_{k-1}^1, a_{k-1}^1) \succ r(o_0^2, a_0^2), ... ,r(o_{k-1}^2, a_{k-1}^2)$

  Generating our **human's preferences** though the use of a **reward function r** should help our agent to receive a high total reward using r. It implies that **knowing r allows us to evaluate/score** the agent quantitatively Ideally we want the agent to have nearly full reward as we would have using RL to find r.

  Qualitative: When there is no **reward function** to **quantitatively evaluate** our trajectories, the approach presented in the article become useful. When it is the case we must **qualitatively evaluate the agent trying to satisfy our humans' expectations**. In the paper they begin with **natural language** purpose asking a human to evaluate the **agent's behavior** on how well the agent fulfills his purpose, and then explain videos of agents trying to fulfill that purpose.

  Their model is based on the **comparisons between trajectories** paving the way for a similar article studying trajectory preference queries used in [12]. The main difference being that **in the article they assume that they can reset the system to an arbitrary state implying that their segments generally begin from different states**.

  Their algorithm avoids the difficulty of interpretation of human comparisons even when they have no human raters experts in this algorithm.

Their method

At each time step their method maintains a policy $\pi : \mathbf{O} \rightarrow \mathbf{A}$ and a reward function estimate $\hat{r} : \mathbf{O} \times \mathbf{A} \rightarrow \mathbf{R}$, both whose parameters are chosen by deep neural networks. These networks are updated by three processes:

1. **The policy** π interacts with the environment and produces a **set of i trajectories** $\tau_1, ..., \tau_i$. They update the parameters of π by traditional RL algorithm in order to maximize the sum of the predicted rewards $\hat{r}_t = (o_t, a_t)$.

2. Then they select pairs of segments σ^1, σ^2 from the trajectories $\tau_1, ..., \tau_i$ produced in step 1, and send them to a human for comparison.

3. **Supervised learning** then helps us **learning** and **optimizing** to fit the comparisons collected from the human at the time t to change the parameters of the mapping $\hat{r}$. They then use those those processes work **asynchronously** with trajectories flowing from process (1) to process (2), human comparisons flowing from process (2) to process (3), and parameters for $\hat{r}$ flowing from process (3) to process (1).

In the next parts they will provide explanations and details.

0.0.1 Policy Optimization

They now have to deal with a **traditional RL problem** after using $\hat{r}$ to compute rewards. Any algorithm appropriate for the domain can be used in RL.

But still they had to notice that the reward function $\hat{r}$ may be non-stationary. That led them to think of methods that are robust to changes in the reward function focusing on policy gradient methods that have already been used successfully for such problems.

In the paper, they use **advantage actor-critic (A2C)** to play Atari games, as an alternative to the asynchronous implementation of A3C. A2C is a **synchronous, deterministic** implementation that **waits for each actor to finish its segment of experience before updating, averaging over all of the actors**. They also use **trust region policy optimization** described as an **iterative procedure for optimizing policies**, with guaranteed monotonic improvement.

By making several approximations to the theoretically-justified procedure, they used a practical algorithm, called **Trust Region Policy Optimization (TRPO)** [10]. to perform simulated robotics tasks. In each case, **they used parameter settings which have been found to work well for traditional RL tasks**.

The only **hyperparameter** that they adjusted was the **entropy bonus for TRPO**. Indeed because TRPO relies on the trust region to ensure adequate exploration, it can lead to **inadequate exploration if the reward function is changing**.

They normalized the rewards produced by $\hat{r}$ to have **zero mean** and **constant standard deviation** which is a typical pre-processing step particularly efficient here since the position of the rewards is **under-determined by our learning problem**.

0.0.2 Preference Elicitation

Here **we give a visualization of two trajectory segments as short movies to our human overseer**. In each experiment, those movies do not length more than 1 or 2 seconds long. We then ask which segment the human preferred or if they are both equally fine or if they are unable to compare them.
The human judgments are recorded in a database D of triples (σ^1, σ^2, μ) where σ^1 and σ^2 are the two segments and μ is a distribution over 1, 2 **indicating which segment the user preferred**.

If the human selects one segment as preferable, then μ puts all of its mass on that choice. If the human marks the segments as equally preferable, then μ is uniform. Finally, if the human marks the segments as **incomparable**, then the comparison is not included in the database.

0.0.3 Fitting the Reward Function

In the article they interpret a **reward function** estimate $\hat{r}$ as a **preference-predictor** treating $\hat{r}$ as a **latent factor** explaining the human's judgments and assume that the human's probability of preferring a segment σ^i **depends exponentially on the value of the latent reward summed over the length of the clip**:

$\hat{P}[\sigma^1 \succ \sigma^2] = exp(\sum \hat{r}(\mathrm{o}^1_t,\, a^1_t))(exp(\sum \hat{r}(\mathrm{o}^1_t,\, a^1_t)) + exp(\sum \hat{r}(\mathrm{o}^2_t,\, a^2_t)))^{-1}$

$\hat{r}$ is chosen to minimize the **cross-entropy loss** between these **predictions** and the actual **human labels**.

$\mathrm{loss}(\hat{r}) = -\sum(\mu(1)\log([\sigma^1 \succ \sigma^2]) + \mu(2)\log([\sigma^2 \succ \sigma^1]))$ on $(\sigma^1,\, \sigma^2,\, \mu) \in D$

They thought of this method following the path of the **Bradley-Terry model** [4] **estimating score functions from pairwise preferences**, and is the specialization of the Luce-Shephard choice rule to **preferences over trajectory segments**.

This **reward function** is equivalent to **give weights** to a preference **ranking** alike the **Elo system 1978** [6] created for chess where the **amount of Elo points** won/lost for every win/lose depends on the **difference between** the Elo points of the **players**. In the case of a tie between the players, the player with less Elo points is likely to win some points after this tie. Indeed **the greater the difference of level between the players the more likely the better player will win the match**.

The **difference in Elo points** of two players allows us to **estimate the probability** of a **player winning** in an analogous way to what they created with the **difference in predicted reward** of two trajectory segments **estimating the probability** that **one is chosen over the other** by the human. Their algorithm takes into account several modifications to this simple approach.

• First they fit an **ensemble of predictors**, each trained on **D triples** sampled from D with replacement. The estimate $\hat{r}$ is then defined by **normalizing every predictors independently** and finally **average the result**.

• They keep **1/e** part of the **dataset** for each predictor to be used as **validation set**. They then use two **regularization** and adjust the **regularization coefficient** to **keep the validation loss between 1.1 and 1.5 times the training loss**. **Dropout** is another form of regularization possible in some tasks.

• Understanding that **the human raters won't always be right to choose the best option** there exists a constant probability of error which **does not converge to 0 as the difference in reward difference becomes extreme**; they make the assumption that **instead of applying a softmax directly** as described in $\hat{P}[\sigma^1 \succ \sigma^2]$ they assume that there is **10% chance that the human responds uniformly at random**.

0.0.4 Choosing Queries

Now they choose their **query preferences** using an approximation of the uncertainty of the **reward function estimator**:

They **sample a great number of pairs of trajectory segments of length k**. Then **use each reward predictor** in their ensemble to **predict which segment will be preferred from each pair**, and finally **select the trajectories for which the predictions have the highest variance across ensemble members**.

It is a very **harsh approximation** and the article reveals that in some tasks it may **lower the performance**. What they wish is to choose the queries based on the **expected value of information of the query**, but does not explore this subject much as it is an even wider spectrum of work.

Experimental Results

They implemented their algorithm in TensorFlow, interface with MuJoCo and the Arcade Learning Environment through the OpenAI Gym [5].

Reinforcement Learning Tasks with Unobserved Rewards

The first experiments they tried to solve were a **great number of classical tasks for deep RL without observing the true reward**. Instead of knowing the goal, **the agent only learns about the purpose of the task by asking a human which trajectory segments is better** between two samples. Their main purpose is then to solve this task in a **reasonable amount of time** using **as few queries as possible**.

In their experiments, **feedback is provided by contractors** who are given a 1-2 sentence description of each task before being asked to **compare several hundred to several thousand pairs of trajectory segments** for that task. The **duration of each segment was between 1 and 2 seconds long**. The **average query was answered by contractors in about 3 to 5 seconds** and so the experiments involving real human feedback required between 30 minutes and 5 hours of human time.

As a comparison they ran at the same time experiments using **synthetic oracles with preferences over trajectories** reflecting exactly the reward in the underlying task. It means that when the agent requests for a comparison, instead of sending the request to a human, they immediately reply by **indicating a preference** for whichever **trajectory segment actually receives a higher reward** in the **underlying task**.

They also tried to compare to the baseline of RL training **using the real reward**. Their aim was not to get better performance but instead to do nearly as well as RL **without access to reward information** relying on much **scarcer feedback**.

Still, those feed backs from real humans **have the potential to better perform than RL** and already did in some tasks because human feedback may give us a **better-shaped reward**.

The details of the experiments are described after, including **model architectures**, **modifications to the environment**, and the RL algorithms used to **optimize the policy**.

0.1 Simulated Robotics

They firstly tackled **eight simulated robotics** tasks, implemented in **MuJoCo**, and included in **OpenAI Gym**. They made small modifications to these tasks to **avoid encoding information about the task in the environment itself**. The **reward functions** are **quadratic** functions of the features in these tasks and are **linear functions of distances**, **positions** and **velocities**. To compare they included a much easier pendulum task as it is representative of the complexity of tasks studied in prior work.

They display their results of training their agent with 700 queries to a human rater, compared to learning from 350, 700, or 1400 **synthetic queries**, and also RL learning from the real reward.

700 labels are enough to nearly match reinforcement learning on all of these tasks. Using **learned reward functions** for training tends to be **less stable** and have **higher variance**, having a comparable mean performance at the same time.

Astoundingly, **by 1400 labels their algorithm performs slightly better than if it had simply been given the true reward**. It may be explained perhaps because the learned **reward function is better shaped** than the reward learning procedure which assigns positive rewards to all behaviors generally implying high reward.

Using **real human feedback** is usually **only slightly less effective than the synthetic feedback**. Depending on the task human feedback ranged from being half as efficient as ground truth feedback to being equally efficient.

For example on the **Ant task**, the human feedback outperformed the synthetic feedback apparently because we asked humans to prefer trajectories where the robot was "standing upright," which proved to be a **useful fact to shape a better reward function**.

0.2 Atari

After that they chose to focus on a **set of seven Atari games** in the Arcade Learning Environment. They first display the results of training their agent with 5,500 queries to a human rater, compared to learning from 350, 700, or 1400 synthetic queries, and also RL learning from the real reward.

Their method is not matching RL for those challenging environments although it shows a consequent learning on most of those tasks and matches or even exceeds RL on some.

Indeed, on games as **BeamRider** and **Pong**, the results using synthetic labels match or are close to RL even with only 3,300 labels. On **Seaquest** and **Qbert** we learn slower using synthetic feedback but still with an almost even performance level of RL. On **SpaceInvaders** and **Breakout** synthetic feedback does not compete with RL, even though the agent improves noticeably, usually completing the first level in **SpaceInvaders** and reaching a score of 20 on **Breakout**, or 50 with enough labels.

Usually real human feedback performs even or a little bit worse on most games than using synthetic feedback with 40% less labels. It may be explained by the errors of the human overseer in labeling, leading to an inconsistency between different contractors labeling the same run. In some cases the uneven rate of labeling by contractors may also lead to labels overly concentrated in narrow parts of the state space.

The article explains that the previous problems may be resolved by future improvements on the pipeline for outsourcing labels. On games as **Qbert**, their method does not seem to work as they are not even able to beat the first level with real human feedback. They explain this by reminding us that the use of short clips in **Qbert** can be confusing and difficult to evaluate leading to confusing results for the agent to learn.

In the end, **Enduro** is also difficult to learn for A3C because of the difficulty of successfully getting other cars through random exploration. It seems difficult to teach with synthetic labels but on the long run human labelers tend to reward any progress towards passing cars, shaping the reward and thus outperforming A3C in this game resulting in results comparable to what we would get with Deep Q Learning (DQN).

0.3 Novel behaviors

Trying their method on more traditional RL tasks helped them understanding how effective their method was even though their ultimate goal in human interaction is to solve tasks without knowing any reward function. Using the same parameters as in the previous experiments, they proved that their algorithm is able to apprehend novel complex behaviors.

Indeed they showed that :

1. Using only 900 queries by less than an hour, they taught the **Hopper robot** to complete a sequence of backflips. The agent mastered his task in performing a backflip, land upright, and repeat.

2. They taught the **Half-Cheetah robot** stand on one leg while moving forward. This

behavior was trained with 800 queries in one hour.

3. Race alongside other cars in **Enduro**. They trained with about 1,300 queries and 4 million frames of interactions with the environment. The agent almost learned to stay exactly even with other moving cars for most of the episode, though it's confused by the background change. For this task they learned those behaviors using feedback from the authors.

0.4 Ablation Studies

To better understand the performance of their algorithm, they now considered several changes:

1. Randomly selecting queries uniformly instead of prioritizing existing ones for which there is disagreement (**random queries**).

2. They tried training only **one unique predictor** instead of an ensemble. In this case, they also **randomly selected the query**, since there is no ensemble set to **estimate disagreement**.

3. Then they tried training only on queries collected at the beginning of training, not during the entire training period (no online queries).

4. They removed the 2 regularization techniques and just use dropout (**no regularization**).

5. On the robotic tasks, they used only trajectory segments of length 1 (no segments).

6. Instead of using comparisons to fit , they considered providing real **oracles total rewards** over trajectory segments and fit to these total rewards using mean squares error (target).

They then displayed the results of **MuJoCo**, and **Atari**. Of particular interest is the poor performance of **offline reward predictor training**. Indeed they found here that the predictor captures only part of the true reward, maximizing this partial reward can lead to strange behavior, which is undesired as measured by the true reward. For example, sometimes training offline at **Pong** guides our agents to **avoid losing but not scoring**; this can lead to very long volleys repeating the same sequence of events, endlessly. This type of behavior suggests that, in general, human feedback needs to be intertwined with RL learning, rather than providing only statistics.

Their main motivation for **eliciting comparisons** rather than **absolute scores** is that they found that it is **much easier** for **humans to provide consistent comparisons than consistent absolute scores**, especially on continuous control tasks and qualitative tasks.

Nevertheless, understanding how **using comparisons affects performance** seems important. For **continuous control tasks**, they found that predicted comparisons performed better than **predicted scores**.

This may be because the scale of the rewards varies greatly, which complicates regression problems that are significantly smoother when they only need prediction comparisons. In the **Atari** task, they avoided these difficulties by changing the reward and effectively **only predicting symbols** even thought it is not suitable for continuous control tasks, since the **relative magnitude of rewards** is important for learning.

Choosing which model works the best is complicated here because in both tasks comparisons and targets had significantly different performance, but **neither consistently outperformed the other**. They also noticed huge performance differences **using single frames rather than clips**. To match the results using single frames they would need to have collected a significant more amount of comparisons.

Globally they discovered that **asking humans to compare longer clips was way more helpful per clip, and way less helpful per frame**. Indeed they found that for short clips it took human raters a while just to **understand the situation**, while for longer clips the evaluation time was roughly linear in the clip length. They tried to choose the shortest clip length for which the evaluation time was linear. In the **Atari** environments, they also found that it was often easier to compare longer clips because they provide more context than single frames.

Experimental Details

In a lot of RL environments there exists **termination conditions** that depend on the behavior of the agent. For example in an episode the agent could die or fall over. They proved that such **termination conditions encode information about the task** even when the **reward function is not observable**. To avoid this narrow source of supervision that may potentially surprise their attempts to learn from human preferences only, they removed all variable-length episodes:

• To model those cases where the robot could fall, in the **Gym** versions of their robotics tasks, when some parameters get outside of a well defined range the episode ends. They replaced these termination conditions by a **penalty** learned by the agent that encourages the parameters to remain in the range.

• In **Atari** games, they did not send life loss or episode end signals to the agent while continuing resetting the environment, successfully **converting the environment into a single continuous episode**. When providing **synthetic oracle feedback** they replaced the ends of an episode by a penalty in all games except **Pong**; the agent must learn this penalty. Taking away episodes' variable length helps leaving the agent with only the information encoded in the environment itself.

In this paradigm **human feedback will become the only provider** for its guidance in learning what we expect the robot to do. In the beginning of the training they compared several trajectory segments drawn from an untrained randomly initialized policy. In the **Atari** domain they also pretrained the reward predictor for 200 epochs before beginning RL training in order to reduce the likelihood of **irreversibly learning a bad policy based on an untrained predictor**.

For the rest of the training, they fed the agent labels at a **rate decaying inversely with the number of timesteps**; after twice as many timesteps have passed. They answer about half as many queries per unit time. This **label annealing** allowed them to balance the importance of having a good predictor from the start with the need to adapt the predictor as the RL agent learns and encounters new states. When training with real human feedback, they attempted to similarly strengthen the label rate, although in practice this is approximate because contractors give feedback at uneven rates. Usually they used an ensemble of 3 predictors, and drew a factor 10 more clip pair candidates than what they finally presented to the human with the presented **clips being selected via maximum variance between the different predictors**.

Simulated Robotics Tasks

The **OpenAI Gym** continuous control tasks penalize large torques. Because torques are not directly visible to a human supervisor, these reward functions are not good representatives of human preferences over trajectories and so they removed them. For the simulated robotics tasks, they optimize policies using **trust region policy optimization** (TRPO) with discount rate $\gamma = 0.995$ and $\lambda = 0.97$.

As **reward predictor** they used a **two layers neural network** with **64 hidden units each**, using **leaky ReLUs** with $\alpha = 0.01$ as nonlinearities. They compared trajectory segments that last 1.5 seconds, depending on the task to achieve that is equivalent to vary between 15 to 60 timesteps. Then they normalized the **reward predictions** to have a **standard deviation of 1**. In the end they also added an **entropy bonus** of 0.01 on every task except swimmer where they used a 0.001 entropy bonus to learn from the reward predictor.

Introducing an **entropy bonus** allowed them to motivate the **increased exploration needed to deal with a changing reward function**.
Finally they chose 25% of their comparisons using a **randomly initialized policy** network at the beginning of training and their rate of labeling after T frames $2 \times 106/(T + 2 \times 106)$.

Atari

They train their **Atari** agents using the standard set of environment wrappers.

At the start of every game, a random number of **no-op actions** are played to introduce variety in the initial game states and to **avoid that the agent memorizes a good sequence of actions from the initial state**. Here they chose 0 to 30 no-ops in the beginning of an episode.

Then they applied **max-pooling** over adjacent frames stacking 4 frames and with a frameskip of 4. If the agent loses its life it will result in the ending of an episode while not resetting the environment and they fixed the rewards between [1, 1].

Atari games include a visual display of the score, which in theory could be used to trivially infer the reward. Since they wanted to focus instead on inferring the reward from the complex dynamics happening in the game, they replaced the score area with a constant black background on all seven games.

On **BeamRider** they additionally blank out the enemy ship count, and on **Enduro** they blank out the speedometer. For the **Atari** tasks they optimized their policies using the A3C algorithm in synchronous form (A2C), using the policy architecture studied in [8]. They also

settled under standard settings for the **hyperparameters**:
Entropy bonus of $\beta = 0.01$
Learning rate of 0.0007 decayed linearly to reach zero after 80 million timesteps.
n = 5 **steps per update**; N = 16 **parallel workers**
Discount rate $\gamma = 0.99$, and **policy gradient** using **Adam** with $\alpha = 0.99$ and $\epsilon = 105$.

For the **reward predictor**, they used 84x84 images as inputs (alike what we have with the inputs to the policy), and stack 4 frames for a total 84x84x4 input tensor.
Then they fed this input through 4 convolutional layers of size 7x7, 5x5, 3x3, and 3x3 with strides 3, 2, 1, 1, each having 16 filters, with **leaky ReLU nonlinearities** ($\alpha = 0.01$).
To complete this approach they then added a fully connected layer of size 64 and with a scalar output.

All convolutional layers use **batch norm** and **dropout** with $\alpha = 0.5$ to prevent overfitting for the predictor. In addition they added **2 regularization** with an **adaptative scheme**. The scale of the predictor is arbitrary since they want the reward predictor to be able to compare two sums over the timesteps. Then they **normalize** it to have a standard deviation of 0.05. This allows the authors to pick the same parameters for the reward predictor as the real reward function instead of changing the **hyperparameters**.

Finally they compared **several trajectory segments** of 25 timesteps (1.7 seconds at 15 fps with frame skipping). They picked 500 comparisons from a **randomly initialized policy network at the beginning of training**, and their **rate of labeling** after T frames of training is decreased every 5×106 frames, to be roughly proportional to $5 \times 106/(T + 5 \times 106)$. They **trained the predictor asynchronously** from the RL agent, and on their hardware they usually proceeded 1 label per 10 RL time steps. They kept a buffer of only the last 3,000 labels and looped over this buffer continuously to make sure that the predictor gives enough weight to new labels when the total number of labels becomes large.

Policy gradients

The particularity of their method is to modify the reward according to human preferences, this method can be applied to reinforcement learning algorithms when they use a reward function. The first step therefore consists in implementing a classic reinforcement learning algorithm. As explained in the article, apart from the rewards, the rest is a traditional deep learning problem, so we can choose to implement a large number of different algorithms. However, as their reward estimation method can be non-stationary, they recommend using methods that are robust to changes in the rewards function. So I decided to do reinforcement learning using the policy gradient method. For the implementation of this method i followed the method of the article of `https://www.janisklaise.com/post/rl-policy-gradients/`.

I will start with a reminder of how this algorithm works. We start by defining $\tau = (s_0, a_0, ..., s_{T-1}, a_{T-1}, s_T)$ which corresponds to the states and actions of T steps. We define $R(s_t, a_t)$ which will correspond to the gain obtained for having performed the action a_t during the state s_t. Over the entire sequence T, we therefore have $R(\tau) := \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$. Our goal is to maximize this expected reward :

$$max_\theta E_{\pi_\theta} R(\tau)$$

. π_θ corresponds to the policy, we generally use a neural network, but if the problem is simpler we can take a logistic regression as we will do in this example. Our objective will be to find these parameters allowing to obtain the policy which maximizes the expected reward. In some problems at each time step the reward is simple to explain but it can become complex in some cases. The advantage of the article is precisely to propose a method making it possible to obtain these rewards on sequences τ for problems that are difficult to explain.

In order to find the best combination of parameters, we can do gradient ascent :

$$\theta \leftarrow \theta + \alpha \nabla_\theta E_{\pi_\theta} R(\tau)$$

. With $\nabla_\theta E_{\pi_\theta} R(\tau)$ the gradient with respect to the parameter and α the learning rate.

We note $P(\tau|\theta)$ the probability of the sequence τ under the policy π_θ.. We can then rewrite the gradient:

$$\begin{aligned}\nabla_\theta E_{\pi_\theta} R(\tau) &= \nabla_\theta \sum_\tau P(\tau|\theta) R(\tau) \\ &= \sum_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\ &= \sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta) R(\tau) \\ &= \sum_\tau P(\tau|\theta) \nabla_\theta \ log P(\tau|\theta) R(\tau) \\ &= E_{\pi_\theta}(\nabla_\theta \ log P(\tau|\theta) R(\tau))\end{aligned} \tag{1}$$

By noting the probability of passing to the state s_{t+1} as a function of the state and the action, $p(s_{t+1}|a_t, s_t)$, and the probability of the trajectory as a function parameters

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

. We can rewrite the gradient of the log probability as:

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

Which finally gives us:

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

. we can estimate this expectation with Monte Carlo sampling as:

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

with L the number of trajectories for on update.

Advantage Actor Critic

To go further we will re-implement a second reinforcement learning method, the Advantage Actor critic (A2C) algorithm, which they use in their paper. The main idea of A2C this algorithm is to use the two basic algorithms of reinforcement learning together, namely Policy based agents, the algorithm explained previously, but also value based algorithm, such as Q-learning. The idea of this algorithm is to train on the one hand a political function which will be fit to return a probability distribution on the actions that the agent can take according to its current state. On the other hand, a value function which makes it possible to measure the expected return for an agent following a particular policy. The policy in this algorithm will be called the actor, it is the one who proposes the actions. The value function is called the critic, it is she who evaluates the actions taken by the actor. These two functions will work together in order to solve the given problem. We generally use a neural network for the actor and the critic.

One probleme of the policy gradients algorithm presented before, is that the reward at the end of the episode is used to update the policy. Consequently, we may conclude that a set of action is very good even if one action in the middle of the episode was very bad. To counteract this problem, a large number of samples are needed to obtain the right policy, which consequently slows down the training time.

$$\nabla\theta = \alpha * \nabla_\theta * (\ log\ \pi_\theta(a_t, s_t)) * R(\tau)$$

the idea of this algorithm will be to solve this problem by no longer modifying the weights at the end of the episode but at the end of each time step.

$$\nabla\theta = \alpha * \nabla_\theta * (\ log\ \pi_\theta(a_t, s_t)) * Q(s_t, a_t)$$

As we no longer wait until the end of the episode to modify the weights, we can no longer use the total reward. This is why we use the critic model. As a reminder, the critic model works like a Q-learning algorithm. It will record all the decisions taken in the past, and make this decision based on the maximum expected future gain. We then see that at the start of learning the critic network will be quite bad and it will gradually learn the right actions.

To better understand this algorithm, let's take the example of two people, one is playing a game (the actor) and the other is watching it (the critic). At first the actor doesn't know what to play so takes random actions. The critic will give him advice on how to play better. Based on these, the actor will modify his playing style (weight). The critic will also learn how to modify these advice in order to improve these each time.

$$\nabla\omega = \beta(R(s,a) + \gamma\hat{q}_\omega(s_{t+1}, a_{t+1}) - \hat{q}_\omega(s_t, a_t))\nabla_\omega\hat{q}_\omega(s_t, a_t)$$

with $TD\_Target = R(s,a) + \gamma\hat{q}_\omega(s_{t+1}, a_{t+1})$ and $TD\_Error = TD\_Target - \hat{q}_\omega(s_t, a_t)$. In the A2C algorithm the Advantage correspond to the TD_Error.

CartPole problem

I chose to test these methods a common problem in reinforcement learning the CartPole problem. The goal of this game is to successfully balance the pole as long as possible. To do this, the system is controlled by applying a force +1 or -1 to the cart. For each action taken the environment return several parameters to estimate, the position of the cart, the velocity of the cart, the angle of the pole and the velocity of the pole (see figure 5).

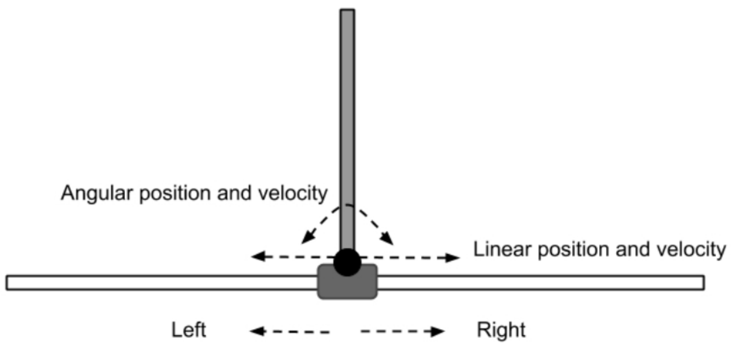


Figure 4: **This figure represents the CartPole problem. We see 4 continuous parameters that we need to find so that the CartPole can be put vertically. Figure taken from the blog :** `https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788629416/9/ch09lvl2sec41/dqn-on-keras`

This problem has the advantage of being able to clearly define the reward function. Indeed at each step where our cart will be above a certain threshold we will give a reward of +1, we stop the sequence if we reach 200 rewards or if the pole falls. Moreover, it is very simple to embark on this challenge because OpenAI ([5]) already provides an implementation of the game. Thus we do not have to re-create the environment, it is enough to give as input the actions that we wish to take in order to to obtain the consequence on our agent as well as the reward. The results with policy optimization are provided in the figure 5, when we take, = 0.002 and = 0.99. The policy used is logistic regression. We clearly see that after only 200 episodes the agent learn to balanced correctly the cart. The second observation that can be made from figure 5 is that we observe a high variance in the results. This is quite typical of this kind of method, the reason comes from the learned policy which can be unstable. One solution could be to increase the number of samples used before modifying the gradient.

For the second implementation I have followed this article `https://towardsdatascience.com/policy-gradient-reinforce-algorithm-with-baseline-e95ace11c1c4`. This algorithm is in the spirit of the A2C presented previously. Indeed, we will also modify R(t) but this time we will remove what we call a baseline $b(s)$ in order to reduce the overall variance.

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

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

This method is very close to the A2C algorithm but it avoids using another model for the estimation of Q(s,a). For baseline learning, it can be learned by reducing the mean squared error of the empirical expected return and the baseline prediction.

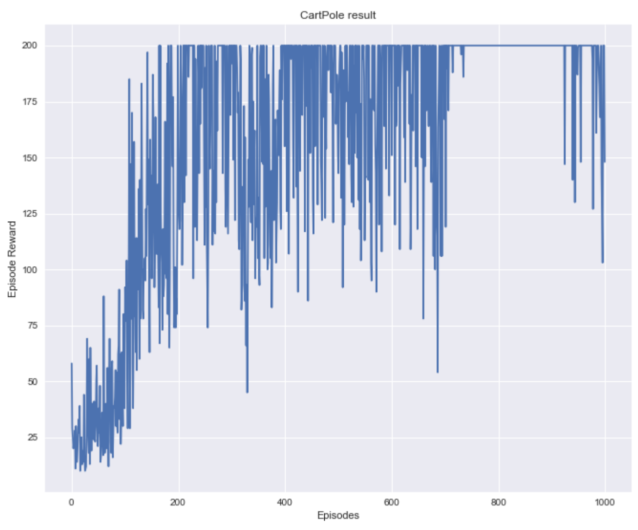


Figure 5: **This figure represents the CartPole results with policy gradient algorithm. We see on the X axis the number of episodes and on the Y axis the number of rewards for this episodes.**

$$MSE = \frac{1}{T}\sum_{t=0}^{T-1}(R(\tau) - b(s_t))^2$$

As can be seen in the 6 the algorithm managed to converge much faster. We note nevertheless that even after convergence at maximum reward, there remain divergences that can be explained by the random aspect of our model.

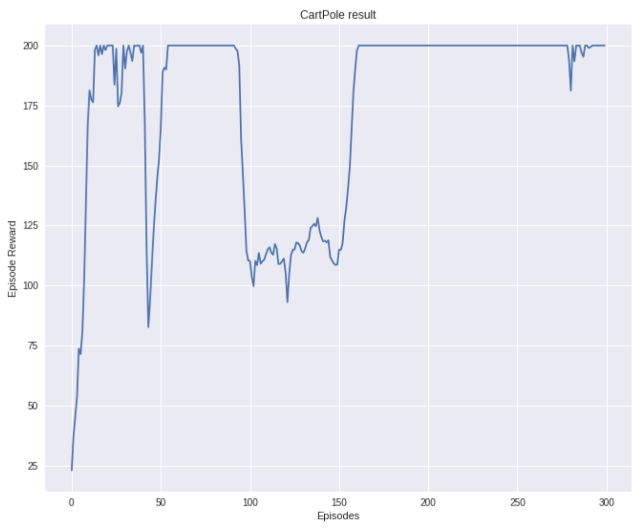
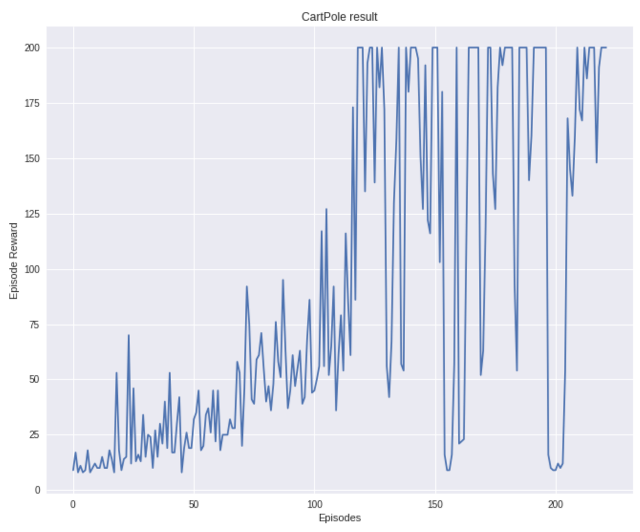


Figure 6: **This figure represents the CartPole results with modified policy optimisation. We see on the X axis the number of episodes and on the Y axis the number of rewards for this episodes.**

For the last implementation I decided to learn how to solve the CartPole problem using the A2C algorithm. I followed the implementation presented in this article `https://github.com/iocfinc/A2C-CartPole`. In 7, We see that, as before even after reaching the maximum score he continues to make mistakes. We explain it by the fact that we still have some stochasticity in our decision-making algorithm. Then, even if the Q-table records the previous best path, this is not always taken. Another observation is that this algorithm takes much longer to train due to these two networks being trained simultaneously. We see also that the A2C algorithm obtain better performances than the policy gradient non modified, but policy gradient modified remain the best.



results.png

Figure 7: **This figure represents the CartPole results with A2C algorithm. We see on the X axis the number of episodes and on the Y axis the number of rewards for this episodes.**

This problem of variance reduction is one of the major problems of RL. There are many methods to mitigate this problem, we have just seen three of them.

Human Preferences rewards

Unfortunately, the authors did not provide their database, with comparisons of several segments. However, this database is difficult to reproduce due to the lack of information on the choice of sequences. I cannot therefore reproduce the results of this article, but I will nevertheless give the steps that should have been followed for the implementation of this method.

In order to modify this rewards, the authors introduced supervised learning to facilitate the learning of the algorithm. They thus created a database containing for each observation, two video sequences of 2 seconds, these sequences then had to be judged by a person who had to say which was the best, that is to say the closest to succeeding the requested action. For our problem, we only have 4 parameters taking continuous values, the velocity of the cart the angle of the pole, Pole angular velocity, cart position. If we want to create sequences of two seconds, we can take 48 images. We can already see one of the problems of this method, it is that the choice of these short sequences covers a huge space of possibility even on a simple problem like this. Unfortunately the authors provide little information on the nature of these video clips which makes it difficult to judge their method. Nevertheless, we know that this database must cover the space of possibilities. Indeed, in a second step they will during the learning step generate K sequences using their algorithm. These sequences must then be compared two by two. However, to do this, they must use supervised learning methods in order to interpolate the response that humans would have had by comparing these multiple sequences, for this interpolation to make sense the sequences must not be too far from the base sequences of training. Finally they will recover the best sequences based on this method. We understand that their method consists in giving a reward when the agent succeeds in the action that we have requested. But by specifying the gain in this way, we are approaching a problem of supervised learning and no longer of reinforcement learning. Indeed, to maximize these gains, the agent will have to get as close as possible to the video sequences that humans consider the best. He will therefore learn to reproduce these video sequences.

Several conclusions on the advantages and disadvantages of this method can be made:
1. The method will greatly accelerate learning.
2. The method is very practical on issues where it is difficult to clearly define what you want.
3. Their method accelerates learning, but leaves less place for the creativity of reinforcement learning algorithms. Indeed, as explained previously, by modifying the reward in this way, the algorithm will learn to reproduce the video sequences, moving from a reinforcement learning problem to a supervised learning problem.
4. The choice of sequences which will have a strong influence on the final result. Moreover, the size of the basis required to cover the space of possibilities will explode if the number of dimensions is increased. Thus, this type of approach can be applied to simple problems but the required database risks exploding if the complexity of the problem is increased. The cost of creating the database is therefore one of the disadvantages of this method.

Discussion and Conclusions

Using agent-environment interactions are often way cheaper to harvest than human interaction. In this article, the authors showed that by learning a separate reward model using supervised learning, it is possible to reduce the interaction complex difficulty by roughly 3. It means that they can usefully train deep RL agents from human preferences, but also that they are fulfilling the task of reducing effort on further sample-complex difficulty improvements because the cost of computation/calculation is already almost the same as the cost of non-expert feedback. They give here one of the first proofs in the literature of RL and elicitation preferences coming from non-specified reward functions that their model can cheaply scale up to the nowadays state-of-the-art RL model. This need to understand the cases where the reward function is unknown is essential for practical applications of deep RL to complex real-world tasks. In the next years, improving the performance of learning from human's mind inclinations will help expanding the range of tasks to which those methods are useful. What the community is nowadays looking at is the ability to make a computer learn a task from human preferences only using a programmatic reward signal, ensuring that powerful RL systems can be applied for completing complex human values tasks instead of easy, low-complexity goals.

References

[1] Riad Akrour, Marc Schoenauer, and Michèle Sebag. “April: Active preference learning-based reinforcement learning”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2012, pp. 116–131.

[2] Riad Akrour et al. “Programming by feedback”. In: *International Conference on Machine Learning*. Vol. 32. JMLR. org. 2014, pp. 1503–1511.

[3] Marc G Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[4] Ralph Allan Bradley and Milton E Terry. “Rank analysis of incomplete block designs: I. The method of paired comparisons”. In: *Biometrika* 39.3/4 (1952), pp. 324–345.

[5] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[6] Arpad E Elo. *The rating of chessplayers, past and present*. BT Batsford Limited, 1978.

[7] Dylan Hadfield-Menell et al. “Cooperative inverse reinforcement learning”. In: *Advances in neural information processing systems* 29 (2016).

[8] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[9] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, p. 2.

[10] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[11] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[12] Aaron Wilson, Alan Fern, and Prasad Tadepalli. “A bayesian approach for policy learning from trajectory preference queries”. In: *Advances in neural information processing systems* 25 (2012).