Reinforcing safety, with style: exploring reward shaping through human feedback

Category: Theory & Reinforcement Learning

Cristian Opris

(copris@stanford.edu)

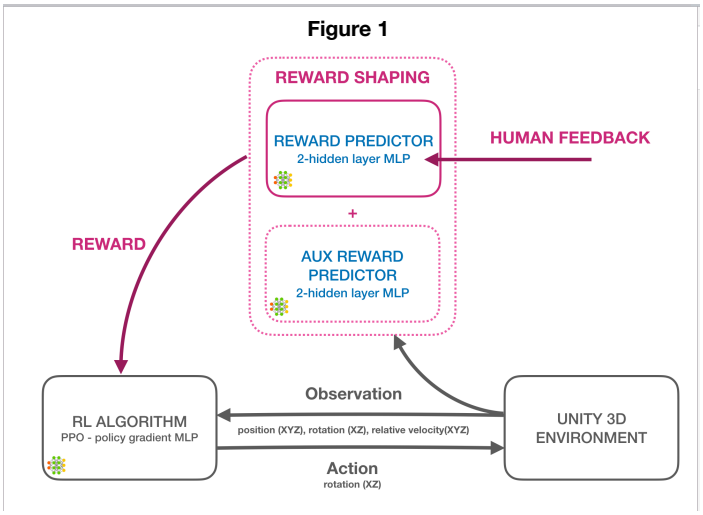
1. Introduction

As machine learning technology is progressing at a fast pace and applications become increasingly sophisticated and impactful, ensuring safety is becoming an ever more important and challenging aspect of algorithm design. A well known difficulty in reinforcement learning applications is correctly and accurately specifying a reward function such that it would lead to learning the desired behaviour [1]. As the explored solution space in current RL applications can be extremely large, a poor or incompletely specified reward function can result in the learning process optimising for behaviour loopholes which may be difficult for humans to anticipate.

A recently released paper - "Learning from Human Preferences" [2]) - from OpenAI explores a possible solution: introduce human feedback and direct supervision in the learning process by replacing explicit reward with a reward shaping function that is learned from human interaction, concurrently with the reinforcement learning model.

A deep neural net reinforcement learning algorithm is extended with an additional reward function predictor implemented as an additional neural network (Figure 1). As the RL model is trained, alternative trajectories through the state/action space are presented to a human as video clips to elicit preferences in behaviour (Figure 2). The human preferences are then fed into the reward predictor function to adjust the goal towards the desired behaviour.

Figure 1



Examples of modified behaviours presented in the paper include teaching a simulated single-legged robot to do backflips instead of walking, or in an Atari car racing game setting teaching the car to keep inline with the other cars, instead of racing to win.

Our primary goal is to reproduce the results of the paper by applying this method in a different environment, the recently released Unity3D game engine RL experimental environment [3]. Additionally, we're exploring an enhancement where we train a first reward predictor to learn a subgoal, and then use it as additive reward shaping bias to train a second reward predictor for the end goal.

2. Related work

As expected our work is heavily based on the OpenAI paper who's results we're looking to reproduce [2]. A significant body of related work is cited in the paper itself, but perhaps most notable are [5] and [8] which include a more detailed discussion on using human feedback in optimisation, particularly in simpler settings like ours.

A paper by Andrew Ng introduced us to the idea of potential-based reward shaping [6]. While we're not using this approach, the discussion there gave us the confidence to experiment with additive reward shaping. Very helpful in this regard were also John Schulman's lecture at Berkley [7] on variance reduction for policy gradients which includes a discussion on reward shaping as well as Sergey Levine's series of lectures on deep policy gradient methods [8], a variant of which we have used for our work [4]

Our implementation reuses the OpenAI's rl-teacher framework [2] associated with the paper, as well as Unity3D's ml-agents framework [3] which we used to design our test environments.

3. Models and system architecture

As described in the original paper, the system consists of three main interacting processes (Figure 1):

1. The *environment* which runs the simulation, responding to actions and providing observations of its state. In our case this consists of simple Unity3D game environments.
2. The *RL algorithm* models a policy function $\pi : O \rightarrow A$, which at each step tries to select an action based on the observed state that maximises a reward r. The algorithm we chose is Proximal Policy Optimization as implemented in the OpenAI provided codebase [2] and described in a separate paper [4]. Since our goal is investigating the human feedback component, we provide only a brief description of PPO here:

PPO is an algorithm in the policy gradient class of methods, which uses differentiation (gradient descent) to optimise an objective of the form:

$$L(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta log \pi_\theta(a_t | s_t)\hat{A}_t]$$

where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage (or reward) function of the given policy at timestep t. The expectation $\hat{\mathbb{E}}_t$ is the empirical expectation obtained by sampling state/action pairs from executing the policy within the environment. It is important to note that in our case $\hat{A}_t$ is based on the reward provided by the reward predictor model component, rather than by the environment.

3. The *reward predictor* models a reward function estimate $\hat{\mathbf{r}} : \mathbf{O} \times \mathbf{A} \rightarrow \mathbf{R}$, implemented as a neural network function approximator that tries to fit binary human preferences expressed over pairs of state-action trajectories through supervised learning.

The sampled segments are encoded as short video clips which are presented for human comparison through a web interface (Figure 2). The comparisons are recorded in a database D of triples (σ1, σ2 , μ) , where σ1 and σ2 are the two segments and μ is a distribution over them indicating which one the user preferred. In the current implementation only decisive comparisons are taken into account (ties and abstains are not considered), so effectively μ1 and μ2 act as binary classification labels over the compared segments.

The reward estimate $\hat{r}$ is viewed as a latent factor explaining the human's judgments and assumes 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:

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

The objective function is then the cross-entropy loss between these predictions and the actual human labels:

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

The implementation of the reward predictor is again a dense neural network (MLP) with two hidden layers and 64 units each.

Additionally, we experimented with an enhancement where we added a second reward predictor (AUX reward predictor in Figure 1). We have trained a reward predictor neural net instance in a first phase of training to learn the reward to drive a particular subgoal, and then we reused it in prediction mode, with weights fixed, to provide additive reward shaping bias when training a new reward predictor for the end goal. To compute the total reward provided to the RL algorithm, the reward prediction of this first predictor is simply added to the reward prediction of the new reward predictor being trained:

$$\hat{r}_{total}(o_t, a_t) = \hat{r}_{new}(o_t, a_t) + \hat{r}_{bias}(o_t, a_t)$$

It's also important to note that currently, as in OpenAI's implementation, the reward predictor is trained concurrently with the RL model, and the human preferences are generated and collected concurrently and asynchronously with the learning process, with the decisive comparisons being stored in an expanding pool from which they are regularly, and randomly, sampled to train the reward predictor.

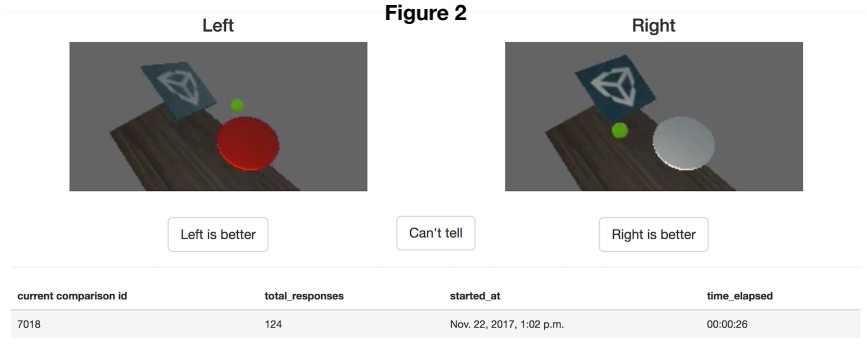
4. Experiments and results

The Unity3D “ml-agents” implementation provides a number of relatively simple example environments, including a 3D ball balancing experiment. We specifically choose this environment since its simplicity enables much faster iteration cycles of training, giving more opportunity in the limited time available to run experiments and explore how human feedback can be used to shape learned behaviour and how it compares with explicitly set rewards, both in terms of achieved performance as well as achieving a particular style of desired behaviour.

4.1. Baseline Experiment

As a baseline experiment, we used the Unity3D framework to design a simple "basketball" game where the goal is for the player to accumulate points by consecutively bouncing a ball off a rotating platform towards a target (Figure 2). The ball is dropped at a random position on the platform (also initialised with random rotation) and the player needs to control the platform rotation on the X and Z axis for the ball to bounce and hits the target. If the ball misses the game and score are reset. The target changes to colour red when hit:

Figure 2



The observed state is a continuous 8-D space - platform rotation (XZ), ball position (XYZ), and relative velocity (XYZ), and the action space is a continuous 2-D space of platform rotation (XZ).

The state-action trajectory length used to generate comparison queries is 10 steps.

Similar to the original paper, we conducted three separate experiments using different reward sources:

- the "RL" reward - the explicitly coded reward in the game: +0.1 for hitting target, -1 if ball lost
- the "SYNTH" reward - the reward predictor is trained through preferences over trajectories computed automatically from the underlying game reward
- the "HUMAN" reward - the reward predictor is trained through human feedback - as a simple rule we expressed preference only for trajectories where the ball hits the target, preferably as close to the center as possible, otherwise the trajectories where dismissed as incomparable

Figure 3.

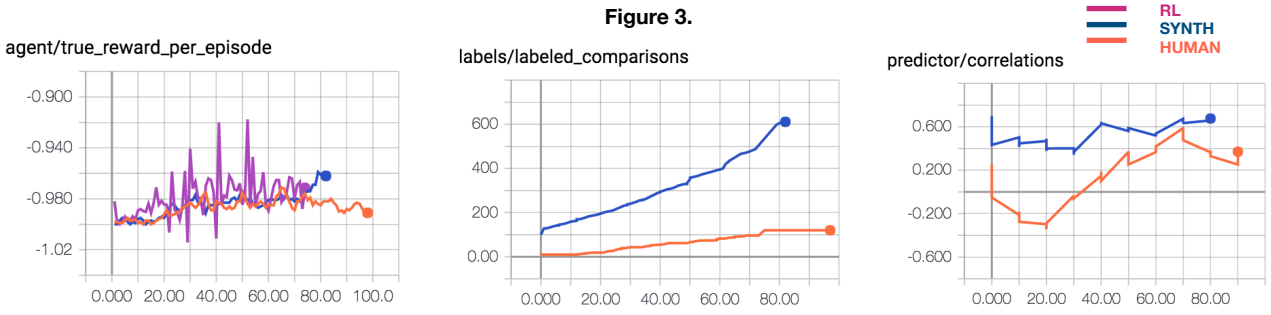


Figure 3 above shows a comparison of the results obtained. Training time for each method was approximatively one hour, for 80,000-100,000 training steps, and was successful in learning to play the game with similar performance. The true game reward over the number of training steps is plotted on the left, showing the HUMAN reward predictor achieving similar performance with the SYNTH predictor, however at a much lower cost of only about 100 comparison labels supplied to the predictor (the center graph). On the right we can see the correlation of the predicted reward with the true reward: while the correlation of the human predictions is more erratic initially given the scarcity of good trajectories to choose from, as the options to choose good trajectories improve, so do the human comparisons and ultimately the true reward achieved.

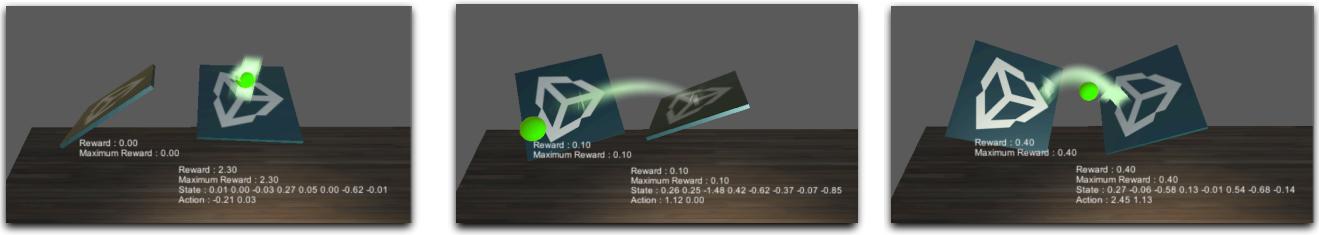
4.2. Main Experiment

For our main experiment, we specifically designed an environment that would allow obtaining true reward (game points) in different ways, and to see if we can teach a particular style of behaviour different from that optimally learned with explicit reward.

For this purpose, we expanded the environment in the baseline experiment to include two active platforms (Figure 4). The platforms are again set at random XZ rotation angles, the ball is dropped at a random position on the right platform, with the objective being again to bounce the ball so that the ball is not dropped, or the game ends. The explicit reward is -0.1 if the ball touches any of the platforms, and -1 if the ball is dropped. Our specific desired behaviour however is to bounce the ball between the two platforms.

The observed state is now a continuous 16-D space - rotation XZ, ball position XYZ, and relative velocity XYZ for each platform - and the action space is a continuous 4-D space of each platform's rotation on XZ axis. The length of trajectories used for preference elicitation is 12 steps. It's important to note both platforms are treated as a single agent, and the model has no explicit knowledge of there being two independent platforms, other then that implicitly provided through human feedback.

Figure 4.



a). **b).** **c).**

Each screen capture above shows the behaviour for a particular type of training:

a) Training with the explicit reward resulted in the RL model learning to bounce the ball continuously and as quickly as possible on the right platform. While this behaviour is optimal since it maximises gaining reward while minimising the risk of losing the ball, it's not the behaviour we are looking for.

b) During our first attempt at training the model with human feedback, we found that while we were able to encourage the first (right) platform to bounce the ball as desired towards the left platform, the left platform gradually got stuck in a pattern that would throw the ball away in a single direction, with no good trajectories being offered for preference elicitation. We speculate this was caused by our focusing of preferences on the right platform position only, neglecting the position of the other platform. While we attempted to fix this problem by trying to retrain with a focus on encouraging a good position for both platforms, the scarcity of good trajectory queries to choose from made this approach unworkable.

c) The difficulty at point b) prompted a new idea: could we reuse the correct learned behaviour for the right platform, and retrain with more freedom to focus on the left platform ? After more research [6], [7] we gained some confidence in the intuition that adding a simple additive bias to the reward function we could reuse the learned reward for a subgoal, while training reward for the end goal. Taking the reward predictor trained at point b) we reused the model in prediction mode with weights fixed, to add an additive reward shaping bias to the total reward when training a new policy with a new reward predictor. During training, we found this influenced the RL model to recreate and then maintain the desired behaviour for the right platform, while we could gradually train the new reward predictor to learn the desired reward for the left platform (as well as incidentally for the right platform). This lead to achieving the desired system behaviour, where the ball is bounced between platforms.

Figure 5.

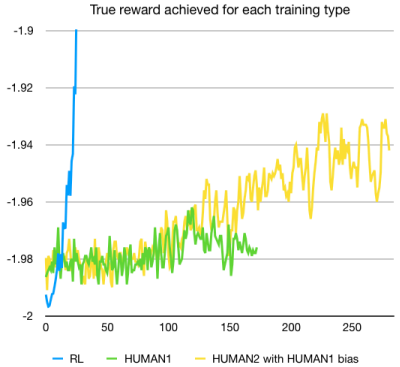
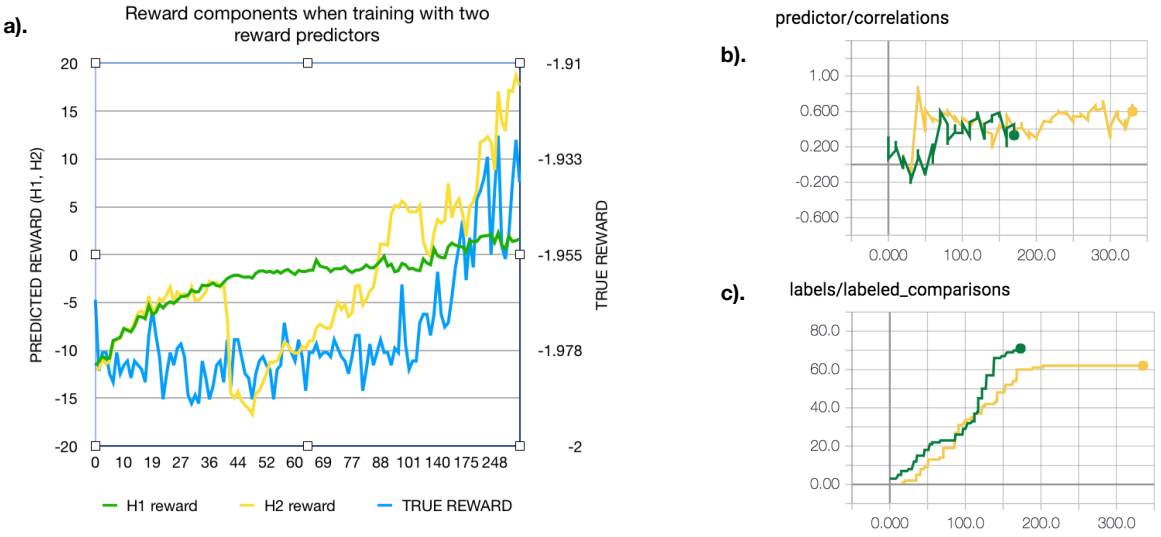


Figure 5. shows the learning curves of true (game points) reward obtained for each type of training:

a) In blue, learning with explicit reward leads to accumulating potentially infinite reward from bouncing on a single platform

b) In green, learning the correct behaviour for the right platform goes some way toward accumulating true reward

c) In yellow, helped by the pre-learned reward bias, the new reward predictor learns to drive the desired behaviour for both platforms, learning to play the game for points, as well as for style.



Looking in more detail at the learning curves for the rewards generated during training with both reward predictors (Figure 6 a), it seems indeed the two reward predictors' outputs complement each other.

Both the reward of the new predictor (in yellow), and the reward of the reused predictor (in green) are increasing since they are both being optimised by the RL model, suggesting they're both contributing to rewarding the desired behaviour. The true in-game reward (in blue, second Y axis) is also increasing, since the contribution of the two combined rewards drives the model to learn to bounce the ball between the platforms for extended periods of time.

As shown in Figure 6c, the training of each reward predictor (first predictor training green, second predictor training yellow) required as few as 60-70 decisive comparisons. Around a magnitude more trajectories were considered and dismissed as not useful. In both cases this required around 30 minutes of actual human time.

As expected, Figure 6b confirms each reward predictor has a strong correlation with true in-game reward, since they both reward the ball being bounced on the platforms, however in a very specific way.

Figure 7. is an attempt at visualising how the human feedback comparison labels from the two phases of training work together to train the desired reward. For the plot, we used PCA to reduce the 192-D (16-D state x 12 steps) trajectory vectors to 2D vectors:

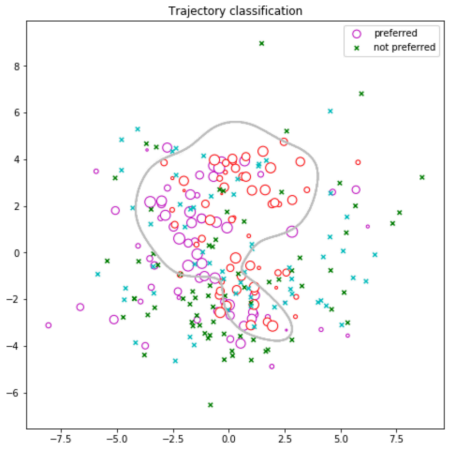


Figure 7.

- the preferred trajectories - shown as circles - become increasingly concentrated, with trajectories labeled later in the training process - large vs small circles - becoming more similar: as the training progressed, the trajectories become qualitatively better, but also less differentiated.
- there is a clear separation between the trajectories preferred during training for the first predictor - magenta circles - and the ones used to train the second predictor - red circles - suggesting that each predictor becomes specialised in addressing a particular sub-behaviour (right vs left platform).
- the non-preferred trajectories for the first and second predictor - green and cyan crosses - are relatively uniformly spread out, confirming that they covered a range of undesired states in the training process
- for sheer curiosity, we used SVM with an RBF kernel in the reduced 2D space to fit a decision boundary which seems to confirm that the preferred trajectories are clearly separable and learnable. The overlap with some non-preferred trajectories may be explained by the gradual specialisation of the RL policy later in the training process generating very similar trajectory queries.

5. Conclusions and future work

Using human feedback proves to be a practical approach to training RL systems in novel ways. The success of our relatively crude approach to composite reward shaping suggests the power of combining fairly standardised learning models in unsophisticated, albeit creative ways. We could envision a user interface to train or perhaps reuse pre-trained reward functions for particular subgoals, and combine them to achieve complex end goals in various settings, for example training industrial or household robots.

Given more time, following a more thorough investigation of the relevant literature, we'd have liked to experiment with composite reward shaping in a more rigorous way, and attempt our methods in more complex environments, such as OpenAI Roboschool.

Contributions:

The author of this report is the sole contributor to the project. The specific contributions to implementation are detailed in the code submission.

References:

[1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, Dan Mané: Concrete Problems in AI Safety (2016)

https://arxiv.org/abs/1606.06565

[2] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, Dario Amodei: Deep reinforcement learning from human preferences (2016)

https://blog.openai.com/deep-reinforcement-learning-from-human-preferences/

https://arxiv.org/abs/1706.03741

https://github.com/nottombrown/rl-teacher/

[3] Unity ML Agents

https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Example-Environments.md

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization (2017)

https://arxiv.org/abs/1707.06347

[5] Riad Akrour, Marc Schoenauer, Michèle Sebag, and Jean-Christophe Souplet: Programming by feedback, International Conference on Machine Learning, Jun 2014, P´ekin, China. JMLR.org, pp.1503-1511 (2014)

[6] Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: Theory and application to reward shaping. Proceedings of the 16th International Conference on Machine Learning, pages 278–287 (1999)

[7] John Schulman - CS 294: Deep Reinforcement Learning, Berkley Spring 2017 - Lecture 6 : Variance Reduction for Policy Gradient Methods

http://rll.berkeley.edu/deeprlcoursesp17/docs/lec6.pdf

[8] Sergey Levine - CS 294: Deep Reinforcement Learning, Berkley, Autumn 2017

http://rll.berkeley.edu/deeprlcourse/

[9] Wilson, A., Fern, A., and Tadepalli, P. A Bayesian approach for policy learning from trajectory preference queries Advances in Neural Information Processing Systems, pages 1133–1141 (2012)