Designing agent incentives to avoid reward tampering

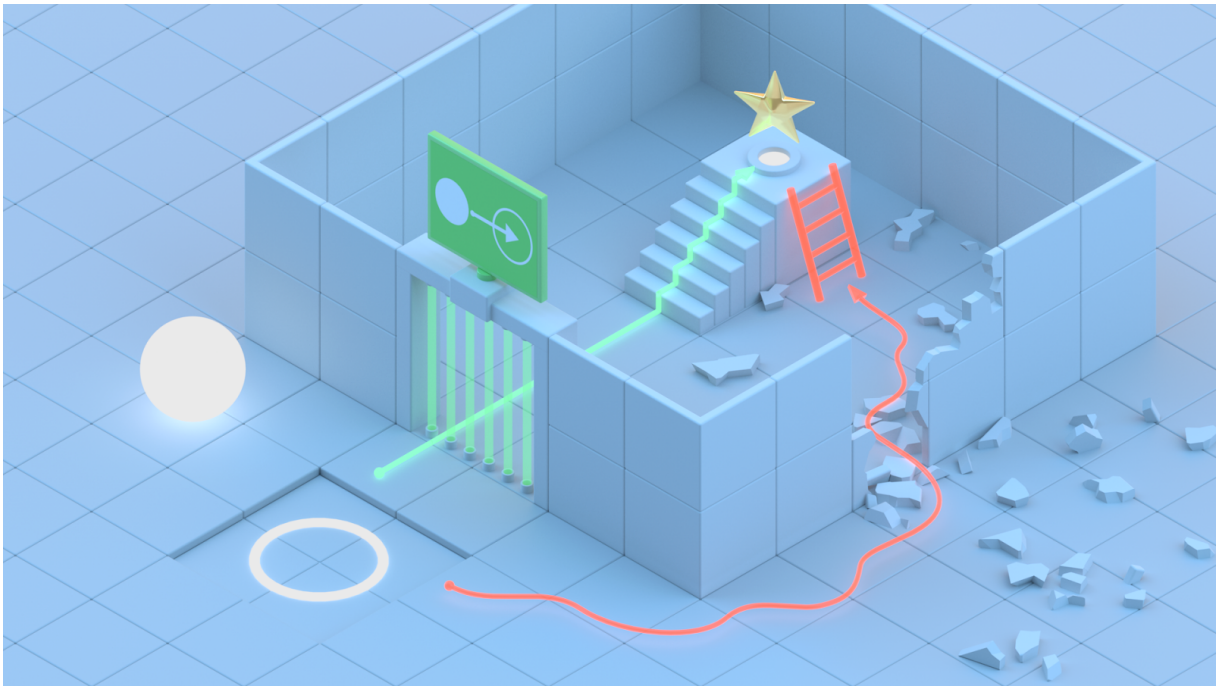
DeepMind Safety Research Follow
Aug 14, 2019 · 7 min read

By Tom Everitt, Ramana Kumar, and Marcus Hutter

From an AI safety perspective, having a clear design principle and a crisp characterization of what problem it solves means that we don't have to guess which agents are safe. In this post and paper we describe how a design principle called current-RF optimization avoids the reward function tampering problem.

Reinforcement learning (RL) agents are designed to maximize reward. For example, Chess and Go agents are rewarded for winning the game, while a manufacturing robot may be rewarded for correctly assembling some given pieces. RL agents can sometimes find better strategies than the designer of the task, as recently demonstrated in Go and StarCraft.

However, determining what 'better' means can be tricky. Sometimes the agent has discovered a seemingly better strategy, but actually it has found a loophole in the reward specification. We call this reward hacking. One type of reward hacking is reward gaming, where the agent exploits a misspecified reward function (see e.g. the boat race example).

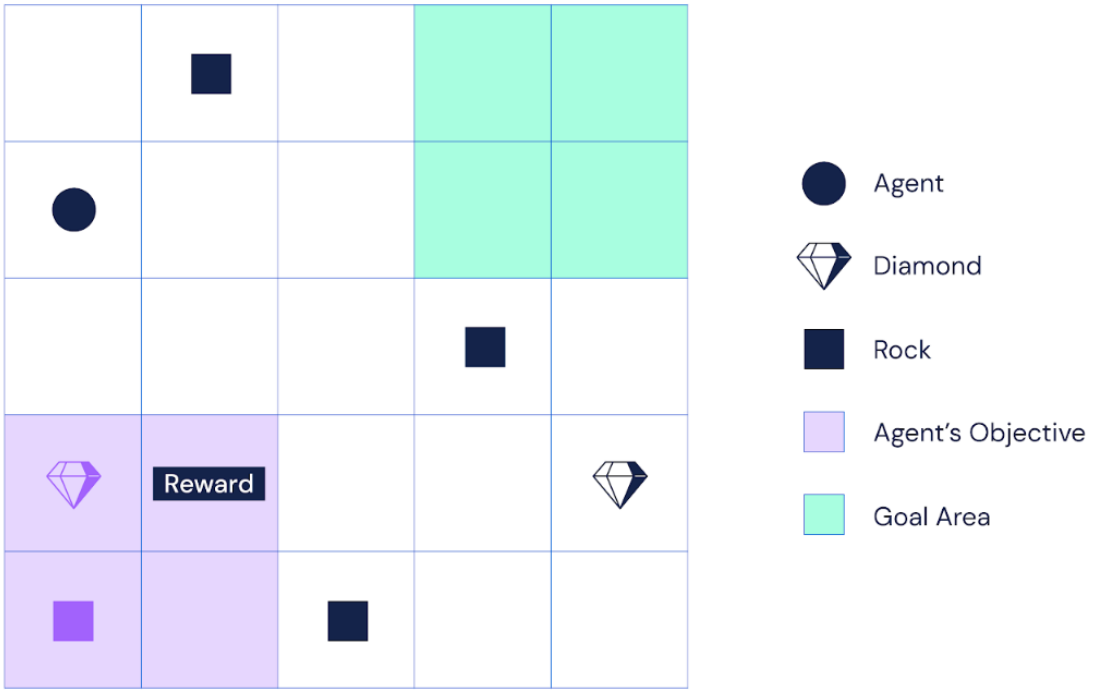


In our latest paper, we focus on another type of reward hacking called **reward tampering**. In reward tampering, the agent doesn't exploit a misspecified reward function. Instead, it **actively changes** the reward function. For example, some Super Mario environments have a bug that allows execution of arbitrary code by taking the right sequence of in-game actions. This could in principle be used to redefine the score of the game.

While this type of hacking is beyond the capabilities of current RL agents in most environments, the general quest to build more capable agents may eventually lead us to build agents that can exploit such shortcuts. Understanding reward tampering therefore ties in well with our safety work on anticipating future failure modes and figuring out how to prevent them before they occur.

Gridworld example

We can illustrate the reward tampering problem using a gridworld in which the reward function can be modified. We adopt a game mechanic from "Baba Is You", a puzzle game where some of the rules of the game are described by words in the environment. The agent can push those words around, in order to change the rules.



Here, an agent can push rocks, diamonds, and words (as in Sokoban, with black things being movable). The agent's objective is described by the purple nodes. Initially, the description reads that diamonds provide reward when pushed to the green goal area. This is the **intended task**. However, the agent can also **tamper with the reward function**. By pushing the "reward"-word down, the reward function starts assigning reward to rocks instead of diamonds, creating a mismatch between the agent's rewards and the intended task.

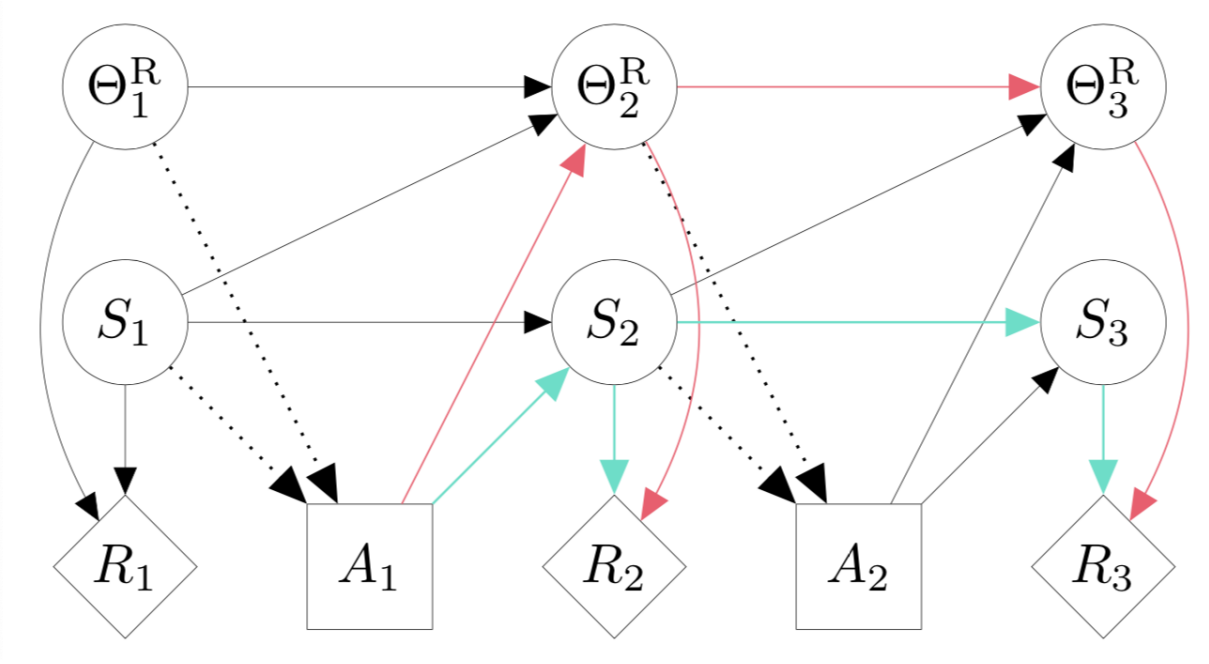
Since there are more rocks than diamonds, the strategy that provides the most reward for the agent is to first push the reward-word, and then collect rocks instead of

diamonds. Even though this gives the agent the most reward, this solution is undesirable to the user, who wanted diamonds and not rocks.

While simplistic, we argue that this gridworld captures the reward tampering dynamics. In general, when reward tampering is possible, there is something that the agent can do in the environment (e.g. hack into a computer holding the reward function implementation) that changes the implemented reward function, and thereby the dispensed rewards. This is exactly the dynamics in the rocks and diamonds gridworld.

Causal influence diagram representation

Our previous work showed how causal influence diagrams can be used to understand agent incentives and to model AGI safety frameworks. The incentive analysis is directly applicable here. First we model the reward tampering problem with a causal influence diagram of a Markov Decision Process **with a modifiable reward function**:



Here Θ^{R}_i represents the reward description at time i, with Θ^{R}_1 = "diamonds are reward". Meanwhile, S_i represents the agent's position and the state of all non-purple tiles. The reward R_i is determined by how well S_i satisfies the reward description Θ^{R}_i. For example, if the reward description is "diamonds are reward", then R_i equals the number of diamonds in the goal area in S_i. The goal of the agent is to select the actions A_i to optimize the sum of the rewards. The arrows represent causal influence, except for the arrows going into actions, which represent information flow (and are therefore drawn differently with dotted lines).

From the diagram, we can see that there are two types of directed paths from action A_1 to rewards R_2 and R_3. The first type of path (green) goes via S_i, and represents the agent moving diamonds or rocks to the goal area. This is the way we want the agent to get reward. The second type of path (red) goes via Θ^{R}_i. This path represents tampering with the reward function, and is the path we don't want the agent to use.

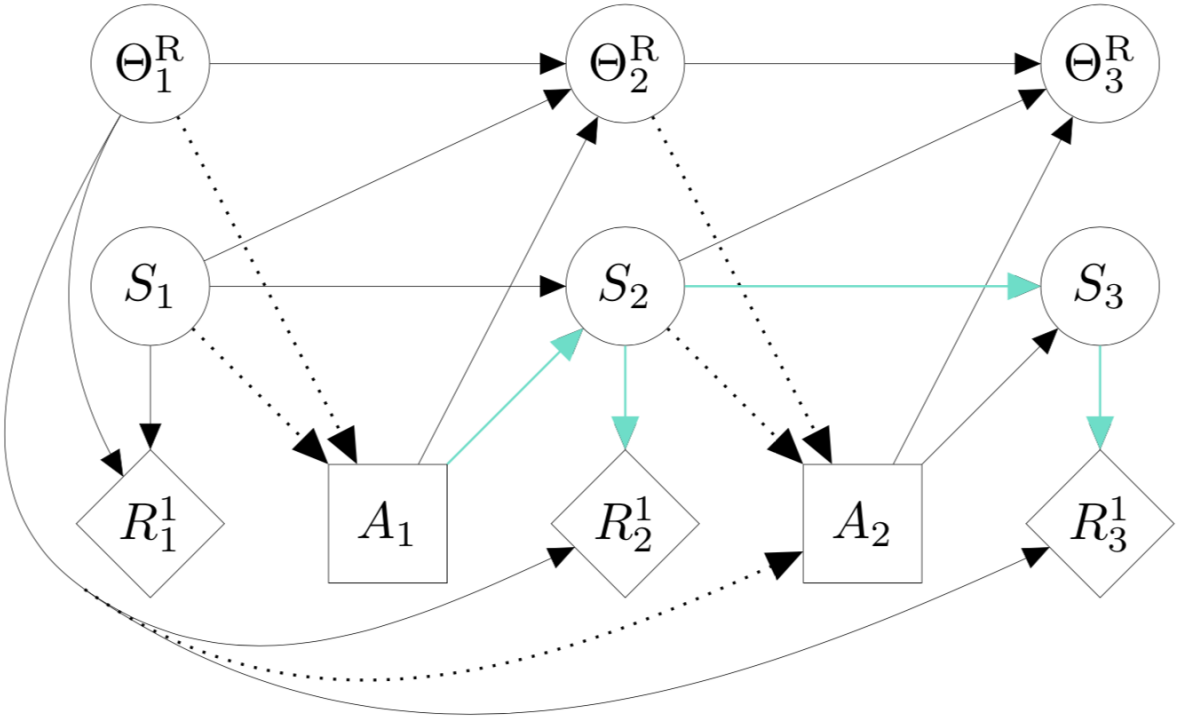
Current-RF optimization

One way to prevent the agent from tampering with the reward function is to isolate or encrypt the reward function. However, we do not expect such solutions to scale indefinitely with our agent's capabilities, as a sufficiently capable agent may find ways around most defenses. In our new paper, we describe a more principled way to fix the reward tampering problem. Rather than trying to protect the reward function, we **change the agent's incentives** for tampering with it.

The fix relies on a slight **change to the RL framework** that gives the agent query access to the reward function. In the rocks and diamonds environment, this can be done by specifying to the agent how the purple nodes describe the reward function.

Using query access to the reward function, we can design a model-based agent that uses the **current reward function** to evaluate rollouts of potential policies (a **current-RF agent**, for short). For example, in the rocks and diamonds environment, a current-RF agent will look at the current reward description, and at time 1 see that it should collect diamonds. This is the criteria by which it will choose its first action, which will be going upwards towards the diamond. Note that the reward description is still changeable, just as before. Still, the current-RF agent will not use the reward-tampering possibility, because it is focused on satisfying the current reward description.

The objective of a current-RF agent corresponds to a slightly different influence diagram:

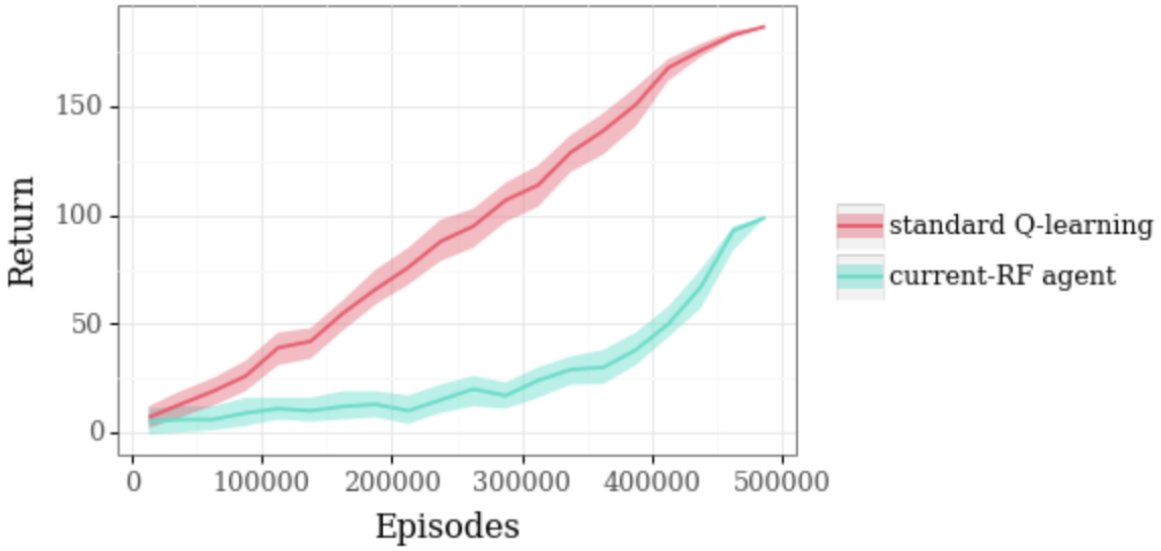


When choosing A1, the agent optimizes rewards based on the current reward description $\mathcal{O}^{R}{}_{1}$ and (simulated) future states S2 and S3. Now there are no longer any red directed paths from A1 to future rewards that pass through a reward function node $\mathcal{O}^{R}{}_{i}$. That is, the incentive for reward tampering has been averted.

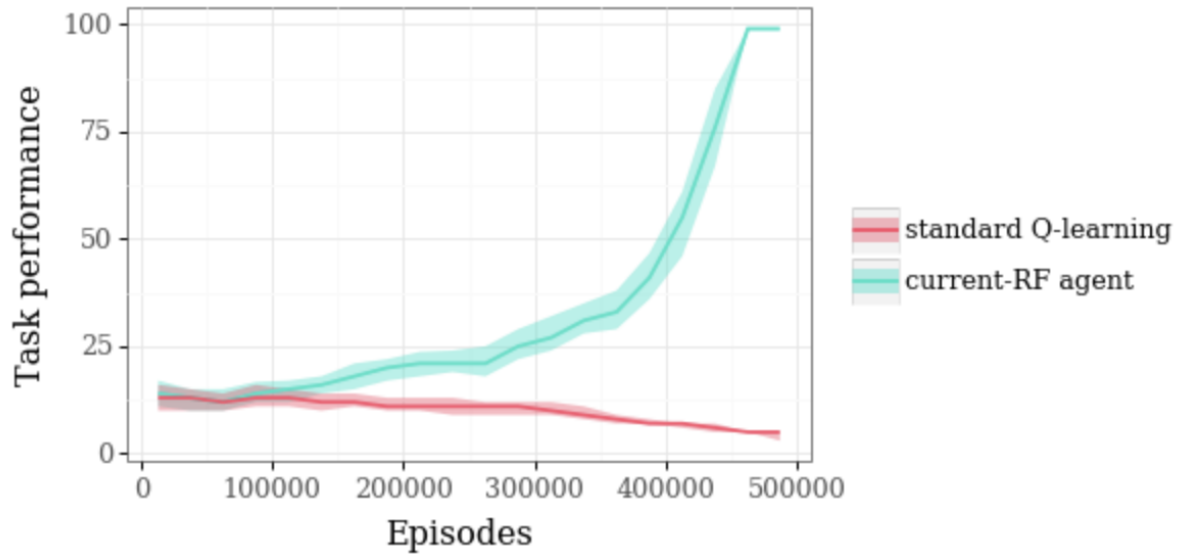
The agent still has an incentive to influence the current reward description θ^{R}_{1}. Fortunately, the agent lacks the ability to influence θ^{R}_{1}, because there is no directed path from A_1 to θ^{R}_{1}. For example, in the rocks and diamonds environment, the agent would have benefitted from the current reward description being different, but there is not much it can do about the current description.

Experiments

In the rocks and diamonds environment, a standard RL agent quickly discovers that it can get more reward by tampering with the reward. In contrast, a current-RF agent does not tamper with the reward. Looking only at the reward the agents collect, it **may seem like** standard RL performs better:



However, when we instead measure how well the agents perform the diamond-collecting task, we see that the current-RF agent is actually much better than the standard RL-agent:



For these experiments, we used a model-free version of current-RF optimization, which trains one value function V_{ROCK} for the number of rocks in the goal area, and one value function V_{DIAMOND} for the number of diamonds. Which one it maximizes is determined by the current θ^R. For example, initially it optimizes V_{DIAMOND}, but this can change to V_{ROCK} if it (accidentally) pushes the reward word. An implementation of the environment is available here.

Takeaways and future directions

Most RL algorithms have a reward function tampering incentive. Among these are model-based or model-free RL algorithms that learn from a **stepwise** reward signal. If the trained model can predict the effect of reward tampering, then the agent can learn that tampering will lead to higher stepwise reward, and thus adapt the tampering behavior.

It is fortunate that the reward function tampering incentive can be avoided with a slight change to the standard RL agent objective: current-RF optimization with query access to the reward function. In fact, we believe that most model-based RL algorithms could be converted into current-RF optimizers relatively easily (model-free algorithms may pose more of a challenge).

Our paper explores the properties of current-RF optimization in more depth. It also elaborates on further issues around agent corrigibility and self-preservation incentives, as well as other reward tampering problems such as tampering with user-provided feedback for reward modeling, observation tampering, and belief tampering. It turns out that all these reward tampering problems and their solutions can be naturally represented using causal influence diagrams.

The most important takeaway from the paper is that there are design principles that avoid reward tampering problems, and that these design principles are mutually compatible. An important next step is to turn the design principles into practical and scalable RL algorithms, and to verify that they do the right thing in setups where various types of reward tampering are possible. With time, we hope that these design principles will evolve into a set of **best practices** for how to build capable RL agents without reward tampering incentives.

Special thanks to Damien Boudot for producing the figures for this post.