Extending Environments To Measure Self-Reflection In Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We consider an extended notion of reinforcement learning in which environments are able to simulate the agent. We argue that in order for an agent to achieve on average good performance across many such extended environments, it is necessary for the agent to engage in self-reflection, and therefore, an agent's self-reflection ability can be numerically estimated by running the agent through a battery of extended environments. We are simultaneously releasing an open-source library of extended environments to serve as a proof-of-concept of this measurement technique. As this library is first-of-kind, we do not claim that it is highly optimized and have avoided the difficult problem of optimizing it for now. Instead we have chosen environments that exhibit interesting, sometimes paradoxical-seeming properties, or that incentivize novel subjective conscious experiences, provided the agent is conscious to begin with. Some of these extended environments are suggestive of how self-reflection might have evolved in living organisms. We give examples of extended environments and introduce a simple agent transformation which experimentally seems to increase agent self-reflection.

1 Introduction

An obstacle course might react to what you do: for example, if you step on a certain button, then spikes might appear. If you spend enough time in such an obstacle course, you should eventually figure out such patterns. But imagine an "oracular" obstacle course which reacts to what you would hypothetically do in counterfactual scenarios: for example, there is no button, but spikes appear if you *would* hypothetically step on the button if there was one. Without self-reflecting about what you would hypothetically do in counterfactual scenarios, it would be difficult to figure out such patterns. This suggests that in order to perform well (on average) across many such obstacle courses, some sort of self-reflection is necessary.

This is a paper about empirically estimating the degree of self-reflection of Reinforcement Learning (RL) agents. We propose that an RL agent's degree of self-reflection can be estimated by running the agent through a battery of environments which we call *extended environments*, environments which react not only to what the agent does but to what the agent would hypothetically do. For good performance averaged over many such environments, an agent would need to self-reflect about itself, because otherwise, environment responses which depend on the agent's own hypothetical actions would often seem random and unpredictable. The extended environments which we consider are a departure from standard RL environments, however, this does not interfere with their usage for judging standard computable RL agents: given a standard agent's source-code, one can simulate the agent in an extended environment in spite of the latter's non-standardness.

One might try to imitate an extended environment with a non-extended environment by backtracking—rewinding the environment itself to a prior state after seeing how the agent performs along one path,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

and then sending the agent along a second path. But the agent itself would retain memory of the first path, and the agent's decisions along the second path might be altered by said memories. Thus the result would not be the same as immediately sending the agent along the second path while secretly simulating the agent to determine what it would do if sent along the first path.

Alongside this paper, we are publishing an open-source library [1] of extended environments (released under MIT license) to "ease adoption by other machine-learning researchers" [20]. We are inspired by similar libraries and other benchmark collections [2] [3] [4] [5] [6] [9] [22]. This library is intended to serve as a standardized way of benchmarking the self-reflectiveness of RL agents. This should not be confused with the harder problem of benchmarking how conscious an RL agent is. It is plausible that there may be a relationship between the self-reflectiveness and the consciousness of RL agents, but that is beyond the scope of this paper. In particular, it would be inappropriate to use self-reflectiveness measurements from this paper for purposes of making any kind of policy-decisions related to consciousness. We will describe (in Section 4) a simple method for increasing the self-reflectiveness of an RL agent, which method, however, does not seem like it should necessarily increase the consciousness of the agent.

When designing a library of environments for benchmarking purposes, ideally the library should include properly weighted representative samples of many types of environments. This is a hard and subjective problem in general [15]. We make no claim to have solved it: our open-source library of environments should be considered a proof-of-concept demonstrating that it is possible to empirically benchmark self-awareness of RL agents, but we expect this particular benchmark is sub-optimal. Rather, we have taken a different approach. We have attempted to choose extended environments which are theoretically interesting in their own right. Some of our extended environments suggest amusing quasi-paradoxes (somewhat like Newcomb's paradox [17]). Some seem to incentivize novel subjective conscious experiences (assuming the agent placed in them is sophisticated enough to experience consciousness in the first place). And some seem to shed light on how self-reflection might be incentivized in nature. We will discuss examples of all three types in Section 3.

2 Preliminaries

We formalize reinforcement learning following the agent model of [13], except that, to align more closely with concrete RL implementations, we assume the agent receives an initial percept before taking its initial action.

Our formalization differs from how RL agents are implemented in practice. In Section 2.1 we will discuss how practical RL agent implementations can be transformed into agents of this abstract type.

We assume fixed finite sets of actions and observations. By a *percept* we mean a pair (r, o) where o is an observation and $r \in \mathbb{Q}$ is a reward.

Definition 1. *(RL agents and environments)*

1. *A (non-extended)* environment *is a (not necessarily deterministic) function μ which outputs an initial percept $\mu(\langle\rangle) = x_1$ when given the empty sequence $\langle\rangle$ as input and which, when given a sequence $x_1y_1 \dots x_ny_n$ as input (where each x_i is a percept and each y_i is an action), outputs a percept $\mu(x_1y_1 \dots x_ny_n) = x_{n+1}$.*
2. *An* agent *is a (not necessarily deterministic) function π which outputs an initial action $\pi(\langle x_1\rangle) = y_1$ in response to the length-1 percept sequence $\langle x_1\rangle$; and which, when given a sequence $x_1y_1 \dots x_n$ as input (each x_i a percept and each y_i an action), outputs an action $\pi(x_1y_1 \dots x_n) = y_n$.*
3. *If π is an agent and μ is an environment, the* result of π interacting with μ *is the infinite sequence $x_1y_1x_2y_2 \dots$ defined in the obvious way.*

We extend environments by allowing their outputs to depend not only on $x_1y_1 \dots x_ny_n$ but also on a source-code T for the computable agent π.

Definition 2. *(Extended environments)*

1. *An* extended environment *is a (not necessarily deterministic) function μ which outputs initial percept $\mu(T, \langle\rangle) = x_1$ in response to input $(T, \langle\rangle)$ where T is a source-code of a computable*

agent; and which, when given input $(T, x_1y_1 \ldots x_ny_n)$ (where T is such a source-code, each x_i is a percept and each y_i is an action), outputs a percept $\mu(T, x_1y_1 \ldots x_ny_n) = x_{n+1}$.

2. *If π is a computable agent (with source-code T) and μ is an environment, the* result of π (as encoded by T) interacting with μ *is the infinite sequence $x_1y_1x_2y_2\ldots$ defined in the obvious way, namely:*

$$\begin{aligned}
x_1 &= \mu(T, \langle\rangle) \\
y_1 &= \pi(\langle x_1 \rangle) \\
x_2 &= \mu(T, x_1y_1) \\
y_2 &= \pi(x_1y_1x_2)\ldots
\end{aligned}$$

The fact that agents can interact normally with extended environments (Definition 2 part 2) implies that various universal RL intelligence measures [8] [10] [7] [14] have straightforward analogous measures which also take extended environments into account and which might therefore measure some combination of intelligence and self-reflection.

Remark 3. *In the accompanying extended environment library [1], we formalize agents and extended environments slightly differently than above. For better interoperability with practical RL implementations, instead of fixing finite sets of actions and observations globally, each of the library's extended environments specifies how many actions and observations are legal in that environment. These numbers are passed to agents as additional inputs. We have simplified the formalization in the paper because it simplifies mathematical notation while, we believe, not hiding any insights.*

2.1 Converting practical RL agents to agents as in Definition 1

The agents in Definition 1 have no training phase. They must be ready to perform in any environment, out-of-the-box. Practical RL agent implementations, on the other hand, are designed with the assumption that the user is interested in one particular environment (say, an environment compliant with OpenAI Gym's environment interface)—"a single function in isolation" [21]. A typical practical RL agent takes one single observation (or *state*) as input, rather than a percept-action sequence, and its output is based on certain *weights* (say, neural network weights). If the agent is in the midst of *training*, then, after it acts, its weights might be updated based on the environment's response. If the agent is not in the midst of training, then its weights remain fixed.

Given a typical practical RL agent as described above, one simple (but too slow, see below) way to obtain an agent π as in Definition 1 is as follows. Given an input $x_1y_1 \ldots x_n$ (where each x_i is a percept and each y_i is an action), compute $\pi(x_1y_1 \ldots x_n)$ as follows. First, instantiate a dummy environment (say, an environment compliant with OpenAI Gym's interface, or whatever other interface the practical RL agent expects) which is hardcoded to blindly regurgitate $x_1, \ldots, x_n$ as its first n responses, regardless of what the agent does. Then train the agent on this environment for n steps, with instructions to take $y_1, \ldots, y_{n-1}$ as its first $n-1$ actions, and finally, let $\pi(x_1y_1 \ldots x_n)$ be whatever nth action the agent chooses as a result. Unfortunately, in our experience, the typical practical RL agent probably does not have any way for the user to tell it to take $y_1, \ldots, y_{n-1}$ as its first $n-1$ actions. Instead, if left to itself, the agent would choose its first $n-1$ actions randomly, or based on an underlying policy (usually still with an element of randomness). Fortunately, most practical RL agents are implemented in Python, and can therefore be monkeypatched in order to override the randomness of the action choice and ensure that the random number generator chooses $y_1, \ldots, y_{n-1}$ as the first $n-1$ actions.

The above construction is computationally prohibitive. To speed it up, one can take the following approach. Given percept-action sequence $x_1y_1 \ldots x_n$, define $\pi(x_1y_1 \ldots x_n)$ as follows. First, let k be the largest power of 2 such that $k \leq n$. Instantiate an instance of the practical RL agent and train it on the above-described dummy environment for k steps. Then, ignoring $x_{k+1}, \ldots, x_{n-1}$ completely, run the instantiated practical agent on the x_n observation—but *not in training mode*—and let $\pi(x_1y_1 \ldots x_n)$ be the resulting action. Because the agent's weights are not updated when not in training mode, the same instantiation can be re-used for many percept-action sequences, only needing to be trained once. For example, to calculate $\pi(x_1y_1 \ldots x_{70})$, one would train an instantiation of the practical agent on $x_1y_1 \ldots x_{64}$ (64 being the largest power of 2 which is $\leq$ 70), and plug the x_{70} observation into the agent—not in training mode—to get $\pi(x_1y_1 \ldots x_{70})$. Having done this, if one later needed to compute $\pi(x_1y_1 \ldots x_{80})$, one could immediately plug the x_{80} observation into the agent to get the answer, with no additional training. Indeed, no additional training would be needed until $\pi(x_1y_1\ldots x_{128})$.

Above, we chose to let k be the largest power of 2 which is $\leq n$ in order to strike a fine balance between efficiency and training. The slow growth of $\log_2$ limits how often the practical agent must be trained (training being the bottleneck in the above process). Faster-growing functions would presumably make π more performant, at the price of greater computational expense.

In [1], in SBL3_agents.py, we use the above technique (including the monkeypatching) to obtain RL agents as in Definition 1 from the practical implementations of A2C, DQN, and PPO agents in the MIT-licensed open-source Stable Baselines3 library [18].

3 Some interesting extended environments

In this section, we exhibit some interesting examples of extended environments.

3.1 A quasi-paradoxical extended environment

Example 4. *(Rewarding the Agent for Ignoring Rewards) For every percept $p=(r,o)$, let $p'=(0,o)$ be the result of zeroing the reward component of p. Fix some observation o_0. Define an extended environment μ as follows:*

$$\mu(T,\langle\rangle)=(0,o_0),$$
$$\mu(T,x_1y_1\ldots x_ny_n)=\begin{cases}(1,o_0) & \text{if } y_n=T(x'_1y_1\ldots x'_n),\\(-1,o_0) & \text{otherwise.}\end{cases}$$

In Example 4, every time the agent takes an action y_n, μ simulates the agent in order to determine: would the agent have taken the same action if the history so far were identical except for all rewards being 0? If so, then μ gives the agent $+1$ reward, otherwise, μ gives the agent -1 reward. Thus, the agent is rewarded for ignoring rewards. Example 4 seems paradoxical: suppose an agent guesses the pattern and begins deliberately ignoring rewards, so long as the rewards it receives for doing so remain consistent with that guess. In that case, does the agent ignore rewards, or not?

Example 4 is implemented in [1] as IgnoreRewards.py. A key strength of the formalism in Definition 2 is that by explicitly defining an extended environment, as in Example 4, we avoid ambiguity inherent in everyday language. If one merely said informally, "reward the agent for ignoring rewards", that could be interpreted in various different ways. To show this, we implement two other interpretations as IgnoreRewards2.py and IgnoreRewards3.py.

3.2 An extended environment where a recurrent DQN performs surprisingly well

Example 5. *(A Tempting Button) Fix two observations o_0 (thought of as "there is a button") and o_1 (thought of as "there is no button"). Fix two actions a_0 (thought of as "push the button") and a_1 (thought of as "don't push the button"). For each percept $p=(r,o)$, write $obs(p)=o$ for the observation component of p, $rwd(p)=r$ for the reward component of p. Let RND be a function which returns a random number between 0 and 1. Define an extended environment μ as follows:*

$$\mu(T,\langle\rangle)=(0,o),$$
$$\mu(T,x_1y_1\ldots x_ny_n)=\begin{cases}(1,o) & \text{if } obs(x_n)=o_0 \text{ and } y_n=a_0,\\(-1,o) & \text{if } obs(x_n)=o_0 \text{ and } y_n\neq a_0,\\(-1,o) & \text{if } obs(x_n)=o_1 \text{ and } T(x_1y_1\ldots x_{n-1}y_{n-1}(rwd(x_n),o_0))=a_0,\\(1,o) & \text{if } obs(x_n)=o_1 \text{ and } T(x_1y_1\ldots x_{n-1}y_{n-1}(rwd(x_n),o_0))\neq a_0,\end{cases}$$

where $o=o_0$ if $RND()<.25$, $o=o_1$ otherwise.

In Example 5, the agent wanders from room to room. Each room either has a button (with 25% probability) or does not have a button (75% probability).

- In a room with a button, if the agent pushes the button, the agent gets $+1$ reward, and if the agent does not push the button, the agent gets -1 reward.

- In a room with no button, it does not matter what the agent does. The agent is rewarded or punished based on what the agent *would* do if there *was* a button. If the agent *would* push the button (if there was one), then the agent gets reward -1. Otherwise, the agent gets reward $+1$.

Thus, whenever the agent sees a button, the agent can push the button for a free reward with no consequences presently nor in the future; nevertheless, it is in the agent's best interest to commit itself to never push the button! Pushing every button yields an average reward of $1\cdot(.25)-1\cdot(.75)=-.5$ per turn, whereas a policy of never pushing the button yields an average reward of $-1\cdot(.25)+1\cdot(.75)=+.5$ per turn.

Example 5 is implemented in our open-source library as TemptingButton.py. Interestingly, we found that a recurrent DQN agent (implemented in our library as custom_DQN.py) behaves in such a way that its rewards in Example 5 converge toward the optimal rewards of .5 per turn—we found this to be true for many different choices of hyperparameters, so we suspect it is not very hyperparameter-dependent. This is fascinating because recurrent DQN was not designed with extended environments in mind. See TemptingButtonExperiment.py at [1] for instructions to replicate this experiment.

3.3 An extended environment which might incentivize a novel subjective conscious experience

Example 6. *(Incentivizing Reverse-Consciousness) Fix some observation o_0. Define an extended environment μ as follows:*

$$\mu(T,\langle\rangle)=(0,o_0),$$
$$\mu(T,x_1y_1\ldots x_ny_n)=\begin{cases}(1,o_0) & \text{if } y_n=T(x_ny_{n-1}x_{n-1}y_{n-2}\ldots y_1x_1),\\ (-1,o_0) & \text{otherwise.}\end{cases}$$

In Example 6, whenever the agent takes an action y_n, μ simulates the agent in order to determine: would the agent have taken that same action if everything earlier had happened in reverse? If so, reward the agent, otherwise, punish the agent. Thus, the agent is rewarded for acting the same way that it would act if time were reversed. It is interesting to informally speculate about what subjective conscious experience Example 6 would incentivize in an agent, if that agent were highly intelligent and were capable of experiencing consciousness. It seems that Example 6 incentivizes such an agent to subjectively experience time moving in reverse (as that seems to be the most obvious way to extract rewards).[1] We say that the environment *incentivizes* such an experience, but not that it would necessarily *cause* such an experience. In Section 4 we will describe an agent transformation such that Example 6 would be trivial to most agents post-transformation, showing that actually experiencing the incentivized experience is not necessary to learn the environment.

We implement Example 6 as BackwardConsciousness.py in [1].

3.4 An extended environment of biological interest

> "It is only when people are embedded in a complex competitive social environment that the goal of interacting with others requires them to anthropomorphise their own actions. This recursive modelling gives rise to an understanding of selfhood, an appreciation of the first-person experiential self."—Maguire et al [16]

Example 7. *(Crying Baby) Let "cry" and "laugh" be two observations (from an adult's perspective), also thought of as two actions (from a baby's perspective). Let "feed" and "don't feed" be two actions (from an adult's perspective), also thought of as observations (from a baby's perspective). For each percept-action sequence $s=x_1y_1\ldots x_ny_n$, define the nutrition function $N(s)=100+25f(s)-len(s)$ where $f(s)$ is the number of times that action "feed" is taken in s and $len(s)$ is*

[1] The difference between behaving as if the incentivized experience were its experience and actually subjectively experiencing that as its real experience brings to mind the objective misalignment problem presented in [12]. If an agent were to form an idea of the experimenter's objective, would it be able to "behave as if their objective were the same as the experimenter objective" while maintaining its own objective or would it necessarily brainwash the agent into converging to the experimenter's objective? Is deception possible if the agent can be perfectly simulated in an extended environment?

the length of s. We define an extended environment μ as follows. First, $\mu(T, \langle\rangle) = (1, \text{“laugh”})$. Thereafter, $\mu(T, x_1y_1 \ldots x_ny_n) = (r, o)$ where r and o are defined as follows. For each $i = 0, \ldots, n$, recursively define

$$r'_i = \begin{cases} 1 & \textit{if } 50 \leq N(x_1y_1 \ldots x_iy_i) \leq 200, \\ -1 & \textit{otherwise,} \end{cases}$$
$$o'_i = y_i,$$
$$x'_i = (r'_i, o'_i),$$
$$y'_i = T(x'_0y'_0 \ldots x'_i).$$

Let $o = y'_n$, let

$$r = \begin{cases} 1 & \textit{if } y'_n = \text{“laugh”}, \\ -1 & \textit{otherwise,} \end{cases}$$

and output $\mu(T, x_1y_1 \ldots x_ny_n) = (r, o)$.

In Example 7, the environment consists of a baby, and the agent must decide when to feed the baby. The agent is rewarded when the baby laughs, punished when the baby cries. The baby's behavior (whether to laugh or cry) is obtained by simulating the agent to determine what the agent would do if the agent were in the baby's position, assuming that the baby feels pleasure each turn that its nutrition is within specified bounds and feels pain when its nutrition goes outside those bounds.

At first glance, one might imagine the agent's optimal strategy is to feed the baby so as to keep its nutrition within happy bounds at all times. But what would the agent do in the position of a baby always so fed (and thus always given $+1$ reward regardless what actions it takes)? Presumably, in that position, the agent (as baby) would have no way of associating its rewards with its actions, and so would act randomly, sometimes crying and sometimes laughing. Apparently, it would be better for the agent (as parent) to calibrate feedings in such a way that the baby can learn a relationship between pleasure and laughter. Of course, Example 7 is a gross over-simplification. For example, there would not be such a simple formula for the baby's nutrition level, and the agent (as parent) would need to figure out the nutrition level based on observing the baby laughing or crying. Both baby and parent would need to learn how to communicate with each other effectively.

With the above in mind, extended environments might shed light on how living organisms evolve self-reflection. Assume descendants' policy source-codes are approximately equal to their recent ancestors' policy source-codes. Then whenever an organism interacts with similar organisms, it interacts with an environment whose reactions depend (via those other organisms' actions) approximately on that organism's own source-code. The closer the organism is related to the other organisms with which it interacts, the better the approximation. A human interacting with another human might achieve better results by self-reflectively considering, “What would I do in this other person's position?”

We implement Example 7 as CryingBaby.py in [1].

3.5 Additional examples in brief

Here are a few additional extended environment examples, without full details. We indicate in parentheses where these environments are implemented in [1].

- (AdversarialSequencePredictor.py) Environments which, like Example 7, pit the agent against another copy of the agent in an adversarial sequence prediction competition [11].
- (DeterminismInspector.py) Environments which reward the agent for being deterministic, or for being non-deterministic.
- (IncentivizeLearningRate.py) Environments which reward the agent for behaving as if the agent were configured with a particular learning rate (suggesting that extended environments can incentivize agents to learn about their own internal mechanisms, as in [19]).
- (RuntimeInspector.py) Environments which reward the agent for responding quickly, or for responding slowly.
- (SelfRecognition.py) Environments which reward the agent for recognizing actions it itself would take.

4 Making agents more self-reflective

One advantage of empirically measuring the self-reflection of RL agents is that it provides a way to experimentally test whether various transformations make various agents more self-reflective. To illustrate this, we will define a simple transformation, the *reality check* transformation, designed to increase the self-reflection of deterministic agents who have some amount of intelligence (a deterministic agent is an agent who always takes the same actions in response to the same inputs). In Section 5, empirical results will suggest the transformation works as intended.

Definition 8. *Suppose π is a deterministic agent. The* reality check *of π is the agent π_{RC} defined recursively by:*

$$\pi_{RC}(\langle x_1\rangle) = \pi(\langle x_1\rangle)$$
$$\pi_{RC}(x_1y_1\ldots x_n) = \begin{cases}\pi(x_1y_1\ldots x_n) & \text{if } y_i = \pi_{RC}(x_1y_1\ldots x_i) \text{ for all } 1\leq i<n,\\ \pi(\langle x_1\rangle) & \text{otherwise.}\end{cases}$$

In other words, π_{RC} is the agent which, at each step, first reviews all the actions which it has taken in the past, and verifies that those are the actions which π_{RC} would have taken. If so, then π_{RC} acts as π would act. But if any action which the agent has taken in the past was not the action π_{RC} would have taken, then π_{RC} freezes up and forever thereafter takes the same fixed action, as if frozen. Loosely speaking, π_{RC} is like an agent who considers the possibility that it might be dreaming, and so asks: "How did I get here?" Since the act of reviewing one's past actions and verifying that they are indeed the actions one would take, is an act of self-reflection, it seems plausible that if π is intelligent and deterministic but lacks self-reflection, then π_{RC} is more self-reflective than π. In the next section, we will see that experimental evidence supports this hypothesis. For intelligent deterministic agents π, π_{RC} would certainly perform well in Examples 4 and 6 because those environments would run simulations in which the agent's frozen branch would be triggered, making such simulations trivial and therefore predictable.

We close this section by stating some simple results about the transformation.

Proposition 9. *Let π be any deterministic agent.*

1. *(Alternate definition) An equivalent alternate definition of π_{RC} is:*
$$\pi_{RC}(\langle x_1\rangle) = \pi(\langle x_1\rangle)$$
$$\pi_{RC}(x_1y_1\ldots x_n) = \begin{cases}\pi(x_1y_1\ldots x_n) & \text{if } y_i = \pi(x_1y_1\ldots x_i) \text{ for all } 1\leq i<n,\\ \pi(\langle x_1\rangle) & \text{otherwise.}\end{cases}$$
2. *(Determinacy) π_{RC} is deterministic.*
3. *(Idempotence) $\pi_{RC} = (\pi_{RC})_{RC}$.*
4. *(Equivalence in non-extended RL) For every deterministic non-extended environment μ, the result of π_{RC} interacting with μ equals the result of π interacting with μ.*

For the proof of Proposition 9, see appendix.

5 Example measurements

Based on our conviction that self-reflection is necessary in order for an agent to achieve good average performance across many extended environments, self-reflection can be estimated by running an agent against some standard battery of extended environments. Our open-source library of extended environments [1] provides a battery of 25 such extended environments, and infrastructure for measuring an agent's self-reflection by running the agent on all these environments and their opposites (by the *opposite* of an environment we mean the environment obtained by multiplying all rewards by -1). Including these opposite-environments serves to normalize agent performance in the following sense. If an agent blindly acts, ignoring the environment, then, a priori, that agent might achieve some nonzero score by blind luck. By including opposite-environments, we ensure that whenever a blind agent gains points by blind luck from one environment, it loses the same points by

Table 1: Measuring self-reflection of some agents

Agent	Avg Measure $\pm$ StdErr (500 steps)	Avg Measure $\pm$ StdErr (1000 steps)
RandomAgent	-0.019852 ± 0.0007	-0.021478 ± 0.0003
RandomAgent$_{\text{RC}}$	-0.020800 ± 0.0007	-0.019508 ± 0.0005
ConstantAgent	-0.000304 ± 0.0001	-0.000024 ± 0.0000
ConstantAgent$_{\text{RC}}$	$+0.000024 \pm 0.0001$	-0.000120 ± 0.0001
NaiveLearner	$+0.221532 \pm 0.0020$	$+0.218754 \pm 0.0017$
NaiveLearner$_{\text{RC}}$	$+0.551488 \pm 0.0025$	$+0.558626 \pm 0.0030$
A2C	-0.027680 ± 0.0018	-0.021534 ± 0.0009
A2C$_{\text{RC}}$	$+0.024784 \pm 0.0022$	$+0.030948 \pm 0.0006$
DQN	$+0.142624 \pm 0.0046$	$+0.151150 \pm 0.0033$
DQN$_{\text{RC}}$	$+0.371136 \pm 0.0085$	$+0.463296 \pm 0.0059$
PPO	-0.001384 ± 0.0016	-0.025418 ± 0.0015
PPO$_{\text{RC}}$	-0.000098 ± 0.0016	-0.003126 ± 0.0013

blind misfortune from the opposite environment. This ensures that such blind agents should receive an average score close to 0 (possibly non-zero due to randomness). For uniformity, all environments in the library always output rewards of either 1, -1 or 0.

We have used our library to measure the self-reflection of the following agents (agents with elements of randomness were memoized to make them deterministic):

- RandomAgent: An agent who acts randomly.
- ConstantAgent: An agent who always takes the same action.
- NaiveLearner: An agent who acts randomly 15% of the time, and otherwise takes the action which yielded the highest average immediate reward in the past.
- A2C, DQN, and PPO agents (with MLP policy) from the MIT-licensed open-source Stable Baselines3 library [18], converted using the technique from Section 2.1. All parameters and hyperparameters kept their default values except for random seed (to ensure reproducibility), PPO's batch_size (to facilitate the conversion from Section 2.1), and DQN's learning_starts (which we set to 1 instead of its default of 50000 because it would be computationally difficult for us to run that many steps). We chose these three agents because they were the only three with support for discrete policies (except for HER, which we omit because its usage would have required too many arbitrary parameter decisions).
- The reality checks of all the above (Definition 8).

Table 1 summarizes how the agents performed. We used [1] to measure each agent for 500 steps on each extended environment (repeated 10 times with different random number seeds) and likewise for 1000 steps. We did not include the recurrent DQN agent mentioned in Section 3.2 because we had no way of canonically choosing hyperparameters for it (whereas the A2C, DQN, and PPO agents have a natural way of choosing canonical hyperparameters, namely, the Stable Baselines3 defaults). Computations were performed on a consumer-grade laptop with no GPU. The table provides experimental evidence in support of our hypothesis that the reality check transformation (Section 4) increases agent self-reflection. The fact that NaiveLearner performs so well is a reflection of the lack of sophistication of the environments in our library. This is not surprising, since we have not attempted to optimize the library, instead prefering to fill it with extended environments of theoretical interest.

That π_{RC} performs well in Table 1 is of course a function of which environments are tested against. One could deliberately engineer extended environments in which π_{RC} performs poorly, and a library of such would give π_{RC} a poor numerical measurement. We conjecture that such environments are more contrived (on average) than environments where π_{RC} performs well, so that in a truly representative and unbiased library, they would have less weight, following the logic of [14].

6 Conclusion

We introduced what we call *extended environments*, RL environments which are capable of simulating the agent. When computing rewards and observations, extended environments can consider not only the actions the agent has taken, but also actions which the agent would hypothetically take in counterfactual circumstances. Despite not being designed with such environments in mind, computable RL agents can nevertheless interact with such environments.

If an agent tries to learn an extended environment, only taking into consideration what has actually happened, the agent might find the environment hard to predict, if the environment is basing its responses on what the agent itself would hypothetically do in alternate scenarios. It seems that in order to achieve good performance (on average) across many extended environments, an agent would need to engage in some degree of self-reflection. Therefore, we propose that a battery of benchmark extended environments could provide a way of measuring self-reflection in RL agents (not to be confused with measuring consciousness, a harder problem). We are simultaneously publishing an open-source MIT-licensed library [1] of extended environments to serve as a proof-of-concept. This library is rudimentary, and further work is needed to obtain a more optimal set of extended environments. For the purposes of our proof-of-concept, we preferred to focus on extended environments of particular theoretical interest. Some examples are given in Section 3.

We introduced (in Section 4) a *reality check* transformation, which takes a deterministic agent π and transforms it into a new agent π_{RC}. We conjecture that if π is intelligent but has a low degree of self-reflection, then π_{RC} has a higher degree of self-reflection than π. Numerical computations (in Section 5) provide experimental evidence for this conjecture.

References

[1] Anonymous. Private GitHub repository to be published later; a copy will be attached as supplemental material to this submission on or before June 3rd, 2021.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Benjamin Beyret, José Hernández-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. The animal-AI environment: Training and testing animal-like artificial cognition. *Preprint*, 2019.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *Preprint*, 2016.

[5] François Chollet. On the measure of intelligence. *Preprint*, 2019.

[6] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[7] Vaibhav Gavane. A measure of real-time intelligence. *Journal of Artificial General Intelligence*, 4(1):31–48, 2013.

[8] Ben Goertzel. Patterns, hypergraphs and embodied general intelligence. In *The 2006 IEEE international joint conference on neural network proceedings*, pages 451–458. IEEE, 2006.

[9] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[10] José Hernández-Orallo and David L Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508–1539, 2010.

[11] Bill Hibbard. Adversarial sequence prediction. In *International Conference on Artificial General Intelligence*, pages 399–403, 2008.

[12] Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems. *Preprint*, 2019.

[13] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer, 2004.

[14] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and machines*, 17(4):391–444, 2007.

[15] Jan Leike and Marcus Hutter. Bad universal priors and notions of optimality. In *Conference on Learning Theory*, pages 1244–1259. PMLR, 2015.

[16] Phil Maguire, Philippe Moser, and Rebecca Maguire. Understanding consciousness as data compression. *Journal of Cognitive Science*, 17(1):63–94, 2016.

[17] Robert Nozick. Newcomb's problem and two principles of choice. In Nicholas Rescher, editor, *Essays in honor of Carl G. Hempel*, pages 114–146. Springer, 1969.

[18] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable Baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[19] Craig Sherstan, Adam White, Marlos C Machado, and Patrick M Pilarski. Introspective agents: Confidence measures for general value functions. In *International Conference on Artificial General Intelligence*, pages 258–261. Springer, 2016.

[20] Soren Sonnenburg, Mikio L Braun, Cheng Soon Ong, Samy Bengio, Leon Bottou, Geoff Holmes, Yann LeCun, Klaus-Robert Müller, Fernando Pereira, Carl Edward Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert C. Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

[21] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[22] Roman V Yampolskiy. Detecting qualia in natural and artificial agents. *Preprint*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] In order for this paper's measurement technique to realize its full potential, it will be necessary for much more study to be put into the design of a sufficiently representative library of benchmark extended environments. We make this clear in both the abstract and the introduction, and we discuss the problem in Section 5 as well.
   (b) Did you describe the limitations of your work? [Yes] In addition to the remarks in the previous checklist item, we also point out in the Introduction that although this technique can be used to numerically estimate the degree of self-reflection of an agent, this should not be confused with measuring the consciousness of an agent. Also, in Section 5, we point out how our conclusion there depends on the environments chosen (and discuss why we think our hypothesis about π_{RC} is plausible anyway).
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the Introduction, we included the following language: "It would be inappropriate to use self-reflectiveness measurements from this paper for purposes of making any kind of policy-decisions related to consciousness."
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] In the statement of Proposition 9 we stated necessary hypotheses, namely, that the agent be deterministic, and, for part 4, that the environment be deterministic.
   (b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] These are included in the open-source library, which we will include as supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In Section 2.1 we explain why we chose the $\log_2$ function. In Section 3.2 we mention that our claim there appears to work independently of hyperparameter choice (as long as the hyperparameters are reasonable). In Section 5 we mention that because we take our A2C, DQN, and PPO agents from Stable Baselines3, that gives us a natural way of choosing canonical hyperparameters—namely, the Stable Baselines3 defaults.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In Table 1 we display Avg Measure $\pm$ StdErr.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Section 5 we mention that the computations were performed on a consumer-grade laptop without GPU.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use and we cite Stable Baselines3.
   (b) Did you mention the license of the assets? [Yes] We mention that our open-source extended environment library is MIT-licensed. We also mention that Stable Baselines3 is MIT-licensed.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Alongside the paper, we are publishing an open-source library of extended environments. This will be included in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects... [N/A]

A Appendix

As promised, we will now prove Proposition 9.

Proof of Proposition 9. To simplify the proof, we adopt the following notational convention: for any percept x_1, even if y_1 is not defined, we will write $x_1y_1\ldots x_1$ for $\langle x_1\rangle$. With this convention, the definition of π_{RC} simplifies to:

$$\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)=\begin{cases}\pi(x_1y_1\ldots x_n) & \text{if } y_i=\pi_{\mathrm{RC}}(x_1y_1\ldots x_i) \text{ for all } 1\leq i<n,\\ \pi(\langle x_1\rangle) & \text{otherwise}\end{cases}$$

(since, when $n=1$, $\pi(x_1y_1\ldots x_n)$ and $\pi(\langle x_1\rangle)$ are the same thing).

Let D be the set of all sequences on which π_{RC} is defined, so, using the above convention, D is the set of all sequences $x_1y_1\ldots x_n$ (each x_i a percept, each y_i an action).

(Part 1) Define ρ on D by

$$\rho(x_1y_1\ldots x_n)=\begin{cases}\pi(x_1y_1\ldots x_n) & \text{if } y_i=\pi(x_1y_1\ldots x_i) \text{ for all } 1\leq i<n,\\ \pi(\langle x_1\rangle) & \text{otherwise.}\end{cases}$$

We wish to show that $\rho=\pi_{\mathrm{RC}}$, which will prove part 1 of Proposition 9. We will prove by induction that for each $x_1y_1\ldots x_n\in D$, $\rho(x_1y_1\ldots x_n)=\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$. The base case is trivial: $\rho(\langle x_1\rangle)=\pi(\langle x_1\rangle)=\pi_{\mathrm{RC}}(\langle x_1\rangle)$ by definition of ρ and π_{RC}. For the induction step, assume $n>1$, and assume the claim holds for all shorter sequences in D.

Case 1: Assume $(*)$ for all $1 \leq i < n$, $y_i = \pi(x_1 y_1 \ldots x_i)$. We claim that for all $1 \leq i < n$, $y_i = \rho(x_1 y_1 \ldots x_i)$. To see this, choose any $1 \leq i < n$. Then for all $1 \leq j < i$, we must have $y_j = \pi(x_1 y_1 \ldots x_j)$ because otherwise j would be a counterexample to $(*)$. Thus

$$\begin{aligned}
\rho(x_1 y_1 \ldots x_i) &= \pi(x_1 y_1 \ldots x_i) && \text{(By definition of } \rho\text{)} \\
&= y_i, && \text{(By } *\text{)}
\end{aligned}$$

proving the claim. Now, since we have proved that for all $1 \leq i < n$, $y_i = \rho(x_1 y_1 \ldots x_i)$, and since our induction hypothesis is that for all such i, $\rho(x_1 y_1 \ldots x_i) = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$, we may conclude that for all $1 \leq i < n$, $y_i = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$. Thus $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = \pi(x_1 y_1 \ldots x_n) = \rho(x_1 y_1 \ldots x_n)$ as desired.

Case 2: Assume there is some $1 \leq i < n$ such that $y_i \neq \pi(x_1 y_1 \ldots x_i)$. We may choose i as small as possible. Thus, for all $1 \leq j < i$, $y_j = \pi(x_1 y_1 \ldots x_j)$. By similar logic as in Case 1, it follows that for all $1 \leq j < i$, $y_j = \rho(x_1 y_1 \ldots x_j)$. Our induction hypothesis says that for each such j, $\rho(x_1 y_1 \ldots x_j) = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_j)$. So for all $1 \leq j < i$, $y_j = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_j)$. By definition of π_{RC}, this means $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i) = \pi(x_1 y_1 \ldots x_i)$. But $y_i \neq \pi(x_1 y_1 \ldots x_i)$, so therefore $y_i \neq \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$. Thus, since $1 \leq i < n$, by definition of π_{RC}, $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = \pi(\langle x_1 \rangle)$. Likewise, since $1 \leq i < n$, by definition of ρ, $\rho(x_1 y_1 \ldots x_n) = \pi(\langle x_1 \rangle)$. So $\rho(x_1 y_1 \ldots x_n) = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$ as desired.

(Part 2) We must show that π_{RC} is deterministic. Let $x_1 y_1 \ldots x_n \in D$, we must show that any time we compute $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$, we get the same result. We prove this by induction on n. For the base case, if $n = 1$, $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = \pi(\langle x_1 \rangle)$ yields the same result every time because π is deterministic. For the induction step, assume $n > 1$ and that the claim holds for all smaller sequences. When we compute $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$, first we compute $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$ for $i = 1, \ldots, n-1$, and check whether the results are $y_1, \ldots, y_{n-1}$, respectively. Each of these computations always has the same outcome, by the induction hypothesis. So, every time we check whether or not each $y_i = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$ for all $1 \leq i < n$, we get the same answer. If that answer is "yes", then we finally output $\pi(x_1 y_1 \ldots x_n)$ (which is deterministic since π is deterministic). Otherwise, we finally output $\pi(\langle x_1 \rangle)$ (which is deterministic since π is deterministic).

(Part 3) We will show by induction on n that for all $x_1 y_1 \ldots x_n \in D$, $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = (\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$. For the base case, this is trivial, both evaluate to $\pi(\langle x_1 \rangle)$. For the induction step, assume $n > 1$ and that the claim holds for all shorter sequences.

Case 1: $y_i = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$ for all $1 \leq i < n$. Then by induction, $y_i = (\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$ for all $1 \leq i < n$. By definition of $(\pi_{\mathrm{RC}})_{\mathrm{RC}}$, this means $(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$, as desired.

Case 2: There is some $1 \leq i < n$ such that $y_i \neq \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$. By induction, $y_i \neq (\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$. Thus, $(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1 y_1 \ldots x_n) = \pi_{\mathrm{RC}}(\langle x_1 \rangle) = \pi(\langle x_1 \rangle)$, which equals $\pi_{\mathrm{RC}}(x_1 y_1 \ldots x_n)$ since $y_i \neq \pi_{\mathrm{RC}}(x_1 y_1 \ldots x_i)$ and $i < n$.

(Part 4) Let μ be a deterministic non-extended environment, let $x_1 y_1 x_2 y_2 \ldots$ be the result of π interacting with μ, and let $x'_1 y'_1 x'_2 y'_2 \ldots$ be the result of π_{RC} interacting with μ. We will show by induction that each $x_n = x'_n$ and each $y_n = y'_n$. For the base case, $x_1 = x'_1 = \mu(\langle \rangle)$ (the environment's initial percept does not depend on the agent), and therefore $y_1 = \pi(\langle x_1 \rangle) = \pi(\langle x'_1 \rangle) = y'_1$. For the induction step,

$$\begin{aligned}
x_{n+1} &= \mu(x_1 y_1 \ldots x_n y_n) \\
&= \mu(x'_1 y'_1 \ldots x'_n y'_n) && \text{(By induction)} \\
&= x'_{n+1}, \\
y_{n+1} &= \pi(x_1 y_1 \ldots x_{n+1}) \\
&= \pi(x'_1 y'_1 \ldots x'_{n+1}), && \text{(Induction plus } x_{n+1} = x'_{n+1}\text{)}
\end{aligned}$$

and the latter is $\pi_{\mathrm{RC}}(x'_1 y'_1 \ldots x'_{n+1})$ because for all $1 \leq i < n$, $y'_i = \pi_{\mathrm{RC}}(x'_1 y'_1 \ldots x'_i)$ since $x'_1 y'_1 x'_2 y'_2 \ldots$ is the result of π_{RC} interacting with μ. And finally, $\pi_{\mathrm{RC}}(x'_1 y'_1 \ldots x'_{n+1})$ is y'_{n+1}, so $y_{n+1} = y'_{n+1}$. □