OpenAI Gym

Greg Brockman, Vicki Cheung, Ludwig Pettersson,
Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba
OpenAI

Abstract

OpenAI Gym[1] is a toolkit for reinforcement learning research. It includes a growing collection of benchmark problems that expose a common interface, and a website where people can share their results and compare the performance of algorithms. This whitepaper discusses the components of OpenAI Gym and the design decisions that went into the software.

1 Introduction

Reinforcement learning (RL) is the branch of machine learning that is concerned with making sequences of decisions. RL has a rich mathematical theory and has found a variety of practical applications [1]. Recent advances that combine deep learning with reinforcement learning have led to a great deal of excitement in the field, as it has become evident that general algorithms such as policy gradients and Q-learning can achieve good performance on difficult problems, without problem-specific engineering [2, 3, 4].

To build on recent progress in reinforcement learning, the research community needs good benchmarks on which to compare algorithms. A variety of benchmarks have been released, such as the Arcade Learning Environment (ALE) [5], which exposed a collection of Atari 2600 games as reinforcement learning problems, and recently the RLLab benchmark for continuous control [6], to which we refer the reader for a survey on other RL benchmarks, including [7, 8, 9, 10, 11]. OpenAI Gym aims to combine the best elements of these previous benchmark collections, in a software package that is maximally convenient and accessible. It includes a diverse collection of tasks (called *environments*) with a common interface, and this collection will grow over time. The environments are versioned in a way that will ensure that results remain meaningful and reproducible as the software is updated.

Alongside the software library, OpenAI Gym has a website (gym.openai.com) where one can find scoreboards for all of the environments, showcasing results submitted by users. Users are encouraged to provide links to source code and detailed instructions on how to reproduce their results.

2 Background

Reinforcement learning assumes that there is an agent that is situated in an environment. Each step, the agent takes an *action*, and it receives an *observation* and *reward* from the environment. An RL algorithm seeks to maximize some measure of the agent's total reward, as the agent interacts with the environment. In the RL literature, the environment is formalized as a partially observable Markov decision process (POMDP) [12].

OpenAI Gym focuses on the episodic setting of reinforcement learning, where the agent's experience is broken down into a series of *episodes*. In each episode, the agent's initial state is randomly sampled from a distribution, and the interaction proceeds until the environment reaches a terminal state. The goal in episodic reinforcement learning is to maximize the expectation of total reward per episode, and to achieve a high level of performance in as few episodes as possible.

The following code snippet shows a single episode with 100 timesteps. It assumes that there is an object called `agent`, which takes in the observation at each timestep, and an object called `env`, which is the

[1] gym.openai.com

environment. OpenAI Gym does not include an agent class or specify what interface the agent should use; we just include an agent here for demonstration purposes.

```
ob0 = env.reset() # sample environment state, return first observation
a0 = agent.act(ob0) # agent chooses first action
ob1, rew0, done0, info0 = env.step(a0) # environment returns observation,
# reward, and boolean flag indicating if the episode is complete.
a1 = agent.act(ob1)
ob2, rew1, done1, info1 = env.step(a1)
...
a99 = agent.act(o99)
ob100, rew99, done99, info2 = env.step(a99)
# done99 == True  =>  terminal
```

3 Design Decisions

The design of OpenAI Gym is based on the authors' experience developing and comparing reinforcement learning algorithms, and our experience using previous benchmark collections. Below, we will summarize some of our design decisions.

Environments, not agents. Two core concepts are the agent and the environment. We have chosen to only provide an abstraction for the environment, not for the agent. This choice was to maximize convenience for users and allow them to implement different styles of agent interface. First, one could imagine an "online learning" style, where the agent takes `(observation, reward, done)` as an input at each timestep and performs learning updates incrementally. In an alternative "batch update" style, a agent is called with observation as input, and the reward information is collected separately by the RL algorithm, and later it is used to compute an update. By only specifying the agent interface, we allow users to write their agents with either of these styles.

Emphasize sample complexity, not just final performance. The performance of an RL algorithm on an environment can be measured along two axes: first, the final performance; second, the amount of time it takes to learn—the sample complexity. To be more specific, final performance refers to the average reward per episode, after learning is complete. Learning time can be measured in multiple ways, one simple scheme is to count the number of episodes before a threshold level of average performance is exceeded. This threshold is chosen per-environment in an ad-hoc way, for example, as 90% of the maximum performance achievable by a very heavily trained agent.
Both final performance and sample complexity are very interesting, however, arbitrary amounts of computation can be used to boost final performance, making it a comparison of computational resources rather than algorithm quality.

Encourage peer review, not competition. The OpenAI Gym website allows users to compare the performance of their algorithms. One of its inspiration is Kaggle, which hosts a set of machine learning contests with leaderboards. However, the aim of the OpenAI Gym scoreboards is not to create a competition, but rather to stimulate the sharing of code and ideas, and to be a meaningful benchmark for assessing different methods.
RL presents new challenges for benchmarking. In the supervised learning setting, performance is measured by prediction accuracy on a test set, where the correct outputs are hidden from contestants. In RL, it's less straightforward to measure generalization performance, except by running the users' code on a collection of unseen environments, which would be computationally expensive. Without a hidden test set, one must check that an algorithm did not "overfit" on the problems it was tested on (for example, through parameter tuning).
We would like to encourage a peer review process for interpreting results submitted by users. Thus, OpenAI Gym asks users to create a *Writeup* describing their algorithm, parameters used, and linking to code. Writeups should allow other users to reproduce the results. With the source code available, it is possible to make a nuanced judgement about whether the algorithm "overfit" to the task at hand.

Strict versioning for environments. If an environment changes, results before and after the change would be incomparable. To avoid this problem, we guarantee than any changes to an environment will be accompanied by an increase in version number. For example, the initial version of the CartPole task is named `Cartpole-v0`, and if its functionality changes, the name will be updated to `Cartpole-v1`.

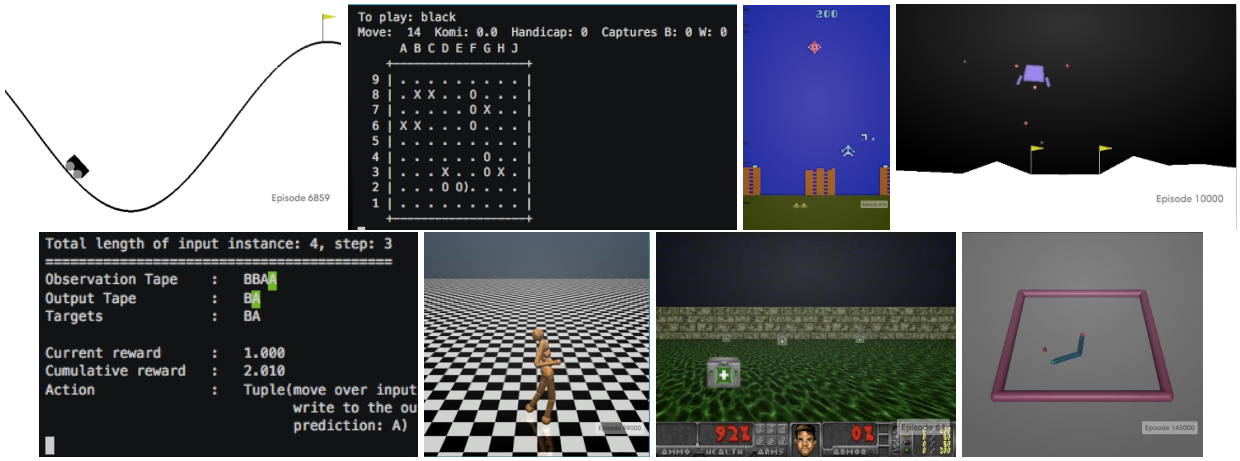

Figure 1: Images of some environments that are currently part of OpenAI Gym.

Monitoring by default. By default, environments are instrumented with a `Monitor`, which keeps track of every time `step` (one step of simulation) and `reset` (sampling a new initial state) are called. The Monitor's behavior is configurable, and it can record a video periodically. It also is sufficient to produce learning curves. The videos and learning curve data can be easily posted to the OpenAI Gym website.

4 Environments

OpenAI Gym contains a collection of Environments (POMDPs), which will grow over time. See Figure 1 for examples. At the time of Gym's initial beta release, the following environments were included:

- *Classic control and toy text*: small-scale tasks from the RL literature.
- *Algorithmic*: perform computations such as adding multi-digit numbers and reversing sequences. Most of these tasks require memory, and their difficulty can be chosen by varying the sequence length.
- *Atari*: classic Atari games, with screen images or RAM as input, using the Arcade Learning Environment [5].
- *Board games*: currently, we have included the game of Go on 9x9 and 19x19 boards, where the Pachi engine [13] serves as an opponent.
- *2D and 3D robots*: control a robot in simulation. These tasks use the MuJoCo physics engine, which was designed for fast and accurate robot simulation [14]. A few of the tasks are adapted from RLLab [6].

Since the initial release, more environments have been created, including ones based on the open source physics engine Box2D or the Doom game engine via VizDoom [15].

5 Future Directions

In the future, we hope to extend OpenAI Gym in several ways.

- *Multi-agent setting*. It will be interesting to eventually include tasks in which agents must collaborate or compete with other agents.
- *Curriculum and transfer learning*. Right now, the tasks are meant to be solved from scratch. Later, it will be more interesting to consider sequences of tasks, so that the algorithm is trained on one task after the other. Here, we will create sequences of increasingly difficult tasks, which are meant to be solved in order.
- *Real-world operation*. Eventually, we would like to integrate the Gym API with robotic hardware, validating reinforcement learning algorithms in the real world.

References

[1] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, Sadik Beattie, C., Antonoglou A., H. I., King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[4] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.

[7] A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How. RLPy: A value-function-based reinforcement learning framework for education and research. *J. Mach. Learn. Res.*, 16:1573–1578, 2015.

[8] B. Tanner and A. White. RL-Glue: Language-independent software for reinforcement-learning experiments. *J. Mach. Learn. Res.*, 10:2133–2136, 2009.

[9] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *J. Mach. Learn. Res.*, 11:743–746, 2010.

[10] S. Abeyruwan. RLLib: Lightweight standard and on/off policy reinforcement learning library (C++). http://web.cs.miami.edu/home/saminda/rilib.html, 2013.

[11] Christos Dimitrakakis, Guangliang Li, and Nikoalos Tziortziotis. The reinforcement learning competition 2014. *AI Magazine*, 35(3):61–65, 2014.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[13] Petr Baudiš and Jean-loup Gailly. Pachi: State of the art open source go program. In *Advances in Computer Games*, pages 24–38. Springer, 2011.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[15] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.