openai / gym

A toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/read-only.html

751 commits | **38** branches | **2** releases | **85** contributors

Branch: **master** | **New pull request** | **Find file** | **Clone or download**

jhseu committed with **wojzaremba** Fix Dict space for list arguments (#713) Latest commit `afe0c42` 9 days ago

bin	Docker - Minor build fix (#286)	a year ago
docs	Fix a typo	6 months ago
examples	Better error when called on non-discrete action spaces	7 months ago
gym	Fix Dict space for list arguments (#713)	9 days ago
misc	TimeLimit refactor with Monitor Simplification (#482)	8 months ago
scripts	minor fixes to rollout generation	4 months ago
vendor	Switch to Docker for tests (#285)	a year ago

.dockerignore	Switch to Docker for tests (#285)	a year ago
.gitignore	Ignore .cache directory created by py-test	a month ago
.travis.yml	comment out docker build line (#710)	13 days ago
CODE_OF_CONDUCT.rst	Initial release. Hello world :).	a year ago
Dockerfile	Allow double-creating the gym directory	a year ago
LICENSE.md	Update LICENSE.md	5 months ago
Makefile	Switch to quay.io/openai/gym:base for tests	a year ago
README.rst	Make sure env.spec always exists and is valid. (#621)	3 months ago
requirements.txt	Pin scipy	a year ago
requirements_dev.txt	switch to pytest (#495)	7 months ago
setup.py	Pin mujoco-py to <1.0.0	3 months ago
test.dockerfile	added ffmpeg to dockerfile and fixed test (#325)	a year ago
tox.ini	Update tox.ini	3 months ago
unittest.cfg	Capture logs in tests	a year ago

README.rst

OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. This is the `gym` open-source library, which gives you access to an ever-growing variety of environments.

build passing

See What's New section below

`gym` makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. You can use it from Python code, and soon from other languages.

If you're not sure where to start, we recommend beginning with the docs on our site. See also the FAQ.

A whitepaper for OpenAI Gym is available at http://arxiv.org/abs/1606.01540, and here's a BibTeX entry that you can use to cite it in a publication:

```
@misc{1606.01540,
        Author = {Greg Brockman and Vicki Cheung and Ludwig Pettersson and Jonas Schneider and John Schulma
        Title = {OpenAI Gym},
        Year = {2016},
        Eprint = {arXiv:1606.01540},
}
```

Contents of this document

- Basics
- Installation
  - Installing everything
  - Supported systems
  - Pip version
  - Rendering on a server
  - Installing dependencies for specific environments
- Environments
  - Algorithmic
  - Atari

    - Board games
    - Box2d
    - Classic control
    - MuJoCo
    - Toy text
- Examples
- Testing
- What's new

Basics

There are two basic concepts in reinforcement learning: the environment (namely, the outside world) and the agent (namely, the algorithm you are writing). The agent sends actions to the environment, and the environment replies with observations and rewards (that is, a score).

The core gym interface is Env, which is the unified environment interface. There is no interface for agents; that part is left to you. The following are the `Env` methods you should know:

- reset(self): Reset the environment's state. Returns observation.
- step(self, action): Step the environment by one timestep. Returns observation, reward, done, info.
- render(self, mode='human', close=False): Render one frame of the environment. The default mode will do something human friendly, such as pop up a window. Passing the close flag signals the renderer to close any such windows.

Installation

You can perform a minimal install of `gym` with:

```
git clone https://github.com/openai/gym.git
cd gym
pip install -e .
```

If you prefer, you can do a minimal install of the packaged version directly from PyPI:

```
pip install gym
```

You'll be able to run a few environments right away:

- algorithmic
- toy_text
- classic_control (you'll need `pyglet` to render though)

We recommend playing with those environments at first, and then later installing the dependencies for the remaining environments.

Installing everything

To install the full set of environments, you'll need to have some system packages installed. We'll build out the list here over time; please let us know what you end up installing on your platform.

On OSX:

```
brew install cmake boost boost-python sdl2 swig wget
```

On Ubuntu 14.04:

```
apt-get install -y python-numpy python-dev cmake zlib1g-dev libjpeg-dev xvfb libav-tools xorg-dev python-op
```

MuJoCo has a proprietary dependency we can't set up for you. Follow the instructions in the `mujoco-py` package for help.

Once you're ready to install everything, run `pip install -e '.[all]'` (or `pip install 'gym[all]'`).

Supported systems

We currently support Linux and OS X running Python 2.7 or 3.5. Some users on OSX + Python3 may need to run

```
brew install boost-python --with-python3
```

If you want to access Gym from languages other than python, we have limited support for non-python frameworks, such as lua/Torch, using the OpenAI Gym HTTP API.

Pip version

To run `pip install -e '.[all]'`, you'll need a semi-recent pip. Please make sure your pip is at least at version `1.5.0`. You can upgrade using the following: `pip install --ignore-installed pip`. Alternatively, you can open setup.py and install the dependencies by hand.

Rendering on a server

If you're trying to render video on a server, you'll need to connect a fake display. The easiest way to do this is by running under `xvfb-run` (on Ubuntu, install the `xvfb` package):

```
xvfb-run -s "-screen 0 1400x900x24" bash
```

Installing dependencies for specific environments

If you'd like to install the dependencies for only specific environments, see setup.py. We maintain the lists of dependencies on a per-environment group basis.

Environments

The code for each environment group is housed in its own subdirectory gym/envs. The specification of each task is in gym/envs/__init__.py. It's worth browsing through both.

Algorithmic

These are a variety of algorithmic tasks, such as learning to copy a sequence.

```
import gym
env = gym.make('Copy-v0')
env.reset()
env.render()
```

Atari

The Atari environments are a variety of Atari video games. If you didn't do the full install, you can install dependencies via `pip install -e '.[atari]'` (you'll need `cmake` installed) and then get started as follow:

```
import gym
env = gym.make('SpaceInvaders-v0')
env.reset()
env.render()
```

This will install `atari-py`, which automatically compiles the Arcade Learning Environment. This can take quite a while (a few minutes on a decent laptop), so just be prepared.

Board games

The board game environments are a variety of board games. If you didn't do the full install, you can install dependencies via `pip install -e '.[board_game]'` (you'll need `cmake` installed) and then get started as follow:

```
import gym
env = gym.make('Go9x9-v0')
env.reset()
env.render()
```

Box2d

Box2d is a 2D physics engine. You can install it via `pip install -e '.[box2d]'` and then get started as follow:

```
import gym
env = gym.make('LunarLander-v2')
env.reset()
env.render()
```

Classic control

These are a variety of classic control tasks, which would appear in a typical reinforcement learning textbook. If you didn't do the full install, you will need to run `pip install -e '.[classic_control]'` to enable rendering. You can get started with them via:

```
import gym
env = gym.make('CartPole-v0')
env.reset()
env.render()
```

MuJoCo

MuJoCo is a physics engine which can do very detailed efficient simulations with contacts. It's not open-source, so you'll have to follow the instructions in mujoco-py to set it up. You'll have to also run `pip install -e '.[mujoco]'` if you didn't do the full install.

```
import gym
env = gym.make('Humanoid-v1')
env.reset()
env.render()
```

Toy text

Toy environments which are text-based. There's no extra dependency to install, so to get started, you can just do:

```
import gym
env = gym.make('FrozenLake-v0')
env.reset()
env.render()
```

Examples

See the `examples` directory.

- Run examples/agents/random_agent.py to run an simple random agent and upload the results to the scoreboard.
- Run examples/agents/cem.py to run an actual learning agent (using the cross-entropy method) and upload the results to the scoreboard.
- Run examples/scripts/list_envs to generate a list of all environments. (You see also just browse the list on our site.

- Run examples/scripts/upload to upload the recorded output from `random_agent.py` or `cem.py`. Make sure to obtain an API key.

Testing

We are using pytest for tests. You can run them via:

```
pytest
```

What's new

- 2017-06-16: Make env.spec into a property to fix a bug that occurs when you try to print out an unregistered Env.
- 2017-05-13: BACKWARDS INCOMPATIBILITY: The Atari environments are now at *v4*. To keep using the old v3 environments, keep gym <= 0.8.2 and atari-py <= 0.0.21. Note that the v4 environments will not give identical results to existing v3 results, although differences are minor. The v4 environments incorporate the latest Arcade Learning Environment (ALE), including several ROM fixes, and now handle loading and saving of the emulator state. While seeds still ensure determinism, the effect of any given seed is not preserved across this upgrade because the random number generator in ALE has changed. The *NoFrameSkip-v4 environments should be considered the canonical Atari environments from now on.
- 2017-03-05: BACKWARDS INCOMPATIBILITY: The configure method has been removed from Env. configure was not used by gym, but was used by some dependent libraries including universe. These libraries will migrate away from the configure method by using wrappers instead. This change is on master and will be released with 0.8.0.
- 2016-12-27: BACKWARDS INCOMPATIBILITY: The gym monitor is now a wrapper. Rather than starting monitoring as env.monitor.start(directory), envs are now wrapped as follows: env = wrappers.Monitor(env, directory). This change is on master and will be released with 0.7.0.
- 2016-11-1: Several experimental changes to how a running monitor interacts with environments. The monitor will now raise an error if reset() is called when the env has not returned done=True. The monitor will only record complete

episodes where done=True. Finally, the monitor no longer calls seed() on the underlying env, nor does it record or upload seed information.

- 2016-10-31: We're experimentally expanding the environment ID format to include an optional username.
- 2016-09-21: Switch the Gym automated logger setup to configure the root logger rather than just the 'gym' logger.
- 2016-08-17: Calling close on an env will also close the monitor and any rendering windows.
- 2016-08-17: The monitor will no longer write manifest files in real-time, unless write_upon_reset=True is passed.
- 2016-05-28: For controlled reproducibility, envs now support seeding (cf #91 and #135). The monitor records which seeds are used. We will soon add seed information to the display on the scoreboard.