10-703 Deep RL and Controls
OpenAI Gym Recitation

Devin Schwab

Spring 2017

Table of Contents

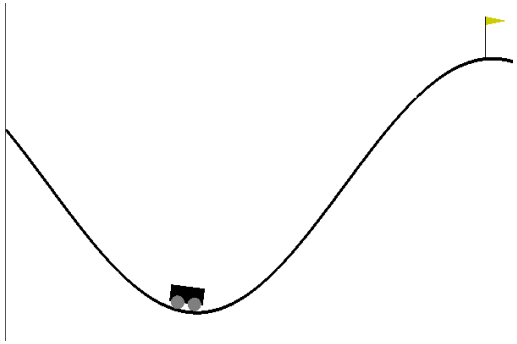
What is OpenAI Gym?

- A standard Python API for RL environments
- A set of tools to measure agent performance
- An online scoreboard for comparing and benchmarking approaches
- `https://gym.openai.com/`

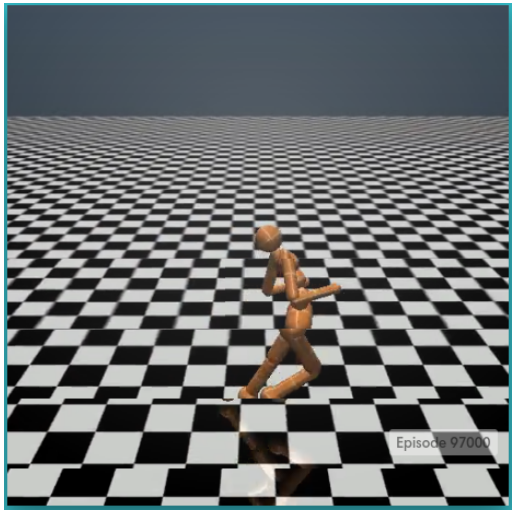
Domain Examples

(a) Toy Text

(b) Atari



(c) Controls



(d) MuJoCo

(e) Doom

(f) Minecraft

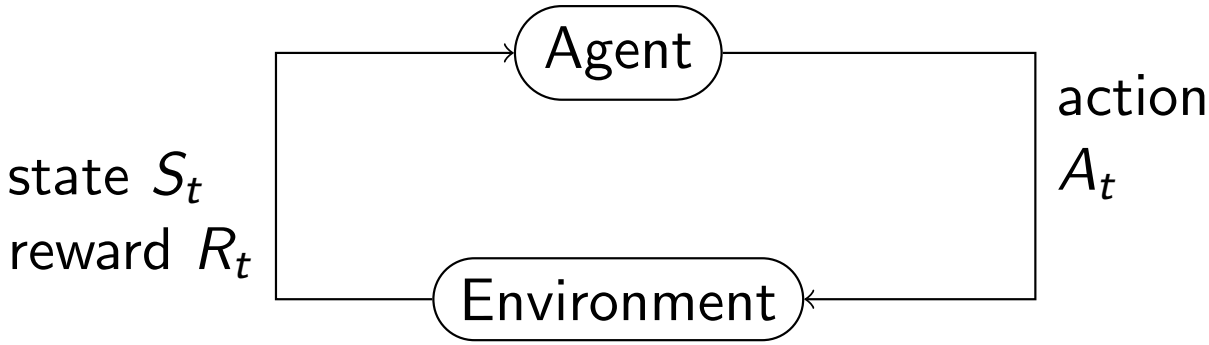
VirtualEnv Installation

- It is recommended that you install the gym and any dependencies in a virtualenv
- The following steps will create a virtualenv with the gym installed

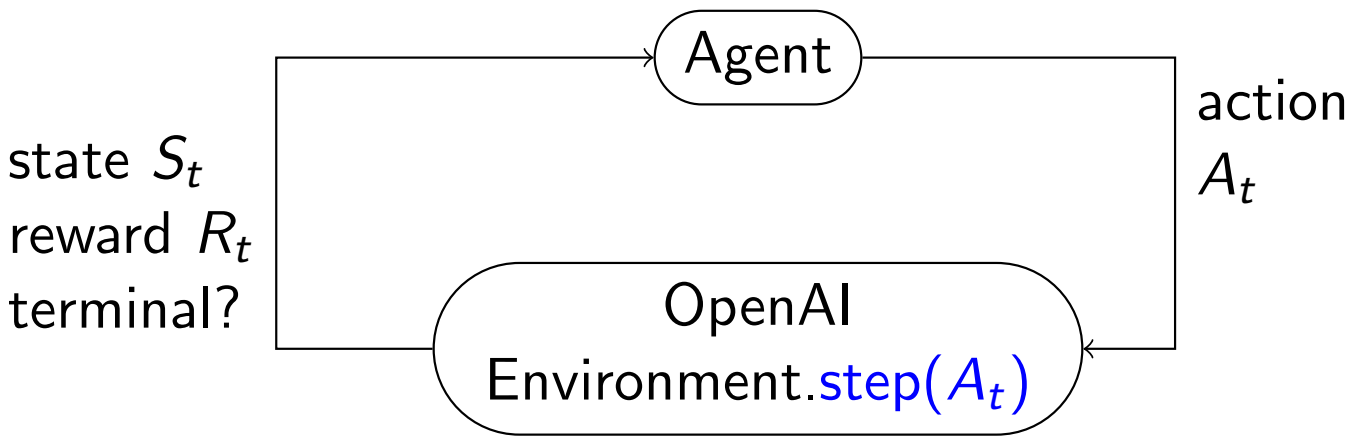
```
virtualenv openai-gym-demo
source openai-gym-demo/bin/activate
pip install -U gym[all]
python -c 'import gym; gym.make("FrozenLake-v0")'
```

Table of Contents

Basic RL Setup



Basic RL Setup



Basic Agent Loop

```
import gym
env = gym.make("Taxi-v2")
observation = env.reset()
for _ in range(1000):
  env.render()
  # your agent here (this takes random actions)
  action = env.action_space.sample()
  observation, reward, done, info = env.step(action)
  if done:
    env.render()
    break
```

Creating an Instance

- Each gym environment has a unique name of the form `([A-Za-z0-9]+-)v([0-9]+)`
- To create an environment from the name use the

```
env = gym.make(env_name)
```

- For example, to create a Taxi environment:

```
env = gym.make('Taxi-v2')
```

Reset Function

- Used to reinitialize a new episode
- Returns the initial state

```
init_state = env.reset()
```

Step Function

```
step(action) -> (next_state,
                 reward,
                 is_terminal,
                 debug_info)
```

- Performs the specified action and returns the resulting state
- The main method your agent interacts with

Render

- Optional method
- Used to display the state of your environment
- Useful for debugging and qualitatively comparing different agent policies

Basic Agent Demo

`demos/basic_agent.py`

Table of Contents

Datatypes

- **Reward** : float
- **Terminal** : bool
- **Action** : Depends on environment
- **State** : Depends on environment

Example State Representations

```
128
```

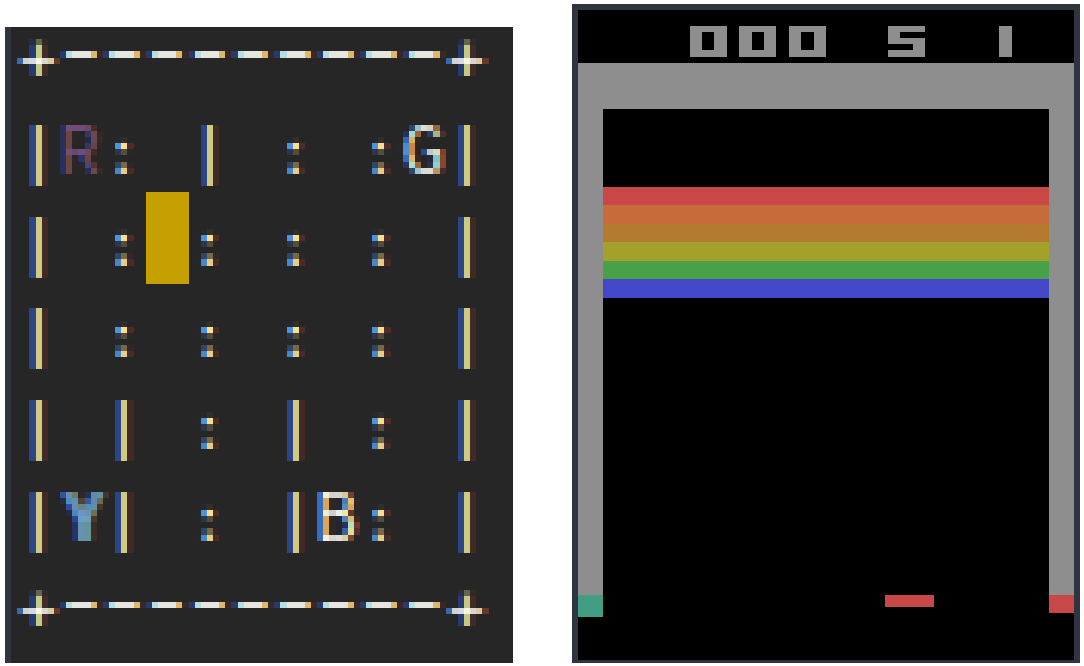
(a) `Taxi-v2`

```
[[[0 0 0]
  [0 0 0]
  [0 0 0]
  ...,
  [0 0 0]
  [0 0 0]
  [0 0 0]]]
```

(b) `Breakout-v0`

Figure: State Representations

Example State Representations



(a) `Taxi-v2` (b) `Breakout-v0`

Figure: State Representations

Example Action Representations

```
1
```

(a) `Taxi-v2`

```
[0, 40.5, 0., -180., .5, 99.2]
```

(b) `Soccer-v0`

Figure: State Representations

Example Action Representations

```
1
```

(a) `Taxi-v2`

```
[0, 40.5, 0., -180., .5, 99.2]
```

(b) `Soccer-v0`

Figure: State Representations

How do you tell what the state and action space is for an environment?

Environment Space Attributes

- Most environments have two special attributes:
  - `action_space`
  - `observation_space`
- These contain instances of `gym.spaces` classes
- Makes it easy to find out what are valid states and actions
- There is a convenient `sample` method to generate uniform random samples in the space.

gym.spaces

- Action spaces and State spaces are defined by instances of classes of the `gym.spaces` modules
- Included types are:
  - `gym.spaces.Discrete`
  - `gym.spaces.MultiDiscrete`
  - `gym.spaces.Box`
  - `gym.spaces.Tuple`
- All instances have a `sample` method which will sample random instances within the space

gym.spaces.Discrete

- The homework environments will use this type of space
- Specifies a space containing n discrete points
- Each point is mapped to an integer from $[0, n-1]$
- `Discrete(10)`
  - A space containing 10 items mapped to integers in $[0, 9]$
  - `sample` will return integers such as 0, 3, and 9.

gym.spaces.MultiDiscrete

- You will use this to implement an environment in the homework
- Species a space containing k dimensions each with a separate number of discrete points.
- Each point in the space is represented by a vector of integers of length k
- `MultiDiscrete([(1, 3), (0, 5)])`
  - A space with $k = 2$ dimensions
  - First dimension has 4 points mapped to integers in $[1, 3]$
  - Second dimension has 6 points mapped to integers in $[0, 5]$
  - `sample` will return a vector such as $[2, 5]$ and $[1, 3]$

gym.spaces.Box

- Used for multidimensional continuous spaces with bounds
- You will see environments with these types of state and action spaces in future homeworks
- `Box(np.array((-1.0, -2.0)), np.array((1.0, 2.0)))`
  - A 2D continous state space
  - First dimension has values in range $[-1.0, 1.0)$
  - Second dimension has values in range $[-2.0, 2.0)$
  - `sample` will return a vector such as $[-.55, 2.]$ and $[.768, -1.55]$

Table of Contents

gym.Env Class

- All environments should inherit from `gym.Env`
- At a minimum you must override a handful of methods:
  - `_step`
  - `_reset`
- At a minimum you must provide the following attributes
  - `action_space`
  - `observation_space`

Subclass Methods

- `_step` is the same api as the step function used in the example
- `_reset` is the same api as the reset function in the example
- You may also provide the following methods for additional functionality:
  - `_render`
  - `_close`
  - `_configure`
  - `_seed`

Attributes

- `observation_space` represents the state space
- `action_space` represents the action space
- Both are instances of `gym.spaces` classes
- You can also provide a `reward_range`, but this defaults to $(-\infty, \infty)$

Registration

- How do you get your environment to work with `gym.make()`?

Registration

- How do you get your environment to work with `gym.make()`?
  - You must register it!

Registration Example

```
from gym.envs.registration import register
register(
    id='Deterministic-4x4-FrozenLake-v0',
    entry_point='gym.envs.toy_text.frozen_lake:FrozenLakeEnv',
    kwargs={'map_name': '4x4',
            'is_slippery': False})
```

Registration Example

- `id` : the environment name used with `gym.make`
- `entry_point` : module path and class name of environment
- `kwargs`: dictionary of keyword arguments to environment constructor

Discrete Environment Class

- A subclass of the `gym.Env` which provides the following attributes
  - `nS` : number of states
  - `nA` : number of actions
  - `P` : model of environment
  - `isd` : initial state distribution

Model

- `P` is a dictionary of dictionary of lists

  `P[s][a] == [(prob, next_state, reward, terminal), ...]`
- `isd` is a list or array of length `nS`

  `isd == [0., 0., 1., 0.]`

FrozenLake-v0 Example

`demos/frozen_lake_demo.py`

Table of Contents

OpenAI Gym Scoreboard

- The gym also includes an online scoreboard
- Gym provides an API to automatically record:
  - learning curves of cumulative reward vs episode number
  - Videos of the agent executing its policy
- You can see other people's solutions and compete for the best scoreboard

Monitor Wrapper

```
import gym
from gym import wrappers
env = gym.make('CartPole-v0')
env = wrappers.Monitor(env, '/tmp/cartpole-experiment-1')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
gym.upload('/tmp/cartpole-experiment-1', api_key='blah')
```

Scoreboard Demo

`demos/monitor_demo.py`

Table of Contents

Summary

- OpenAI Gym provides a standardized API for RL environments
- Gym also provides an online scoreboard for sharing and comparing results/techniques
- With only a few functions you can have your own gym environment to use with your RL algorithms

Thank You

Questions