OpenAI Gym 入门与提高（一） Gym 环境构建与最简单的 RL agent

By Sujim（sjm.nudt@gmail.com）
2016.06.18

Openai gym 是一个用于开发和比较 RL 算法的工具包，与其他的数值计算库兼容，如 tensorflow 或者 theano 库。现在主要支持的是 **python 语言**，以后将支持其他语言。gym 文档在 https://gym.openai.com/docs。

Openai gym 包含 2 部分：

1、**gym 开源库：**包含一个测试问题集，每个问题成为环境（environment），可以用于自己的 RL 算法开发。这些环境有共享的接口，允许用户设计通用的算法。其包含了 deep mind 使用的 Atari 游戏测试床。

2、**Openai gym 服务：**提供一个站点和 api 允许用户对他们训练的算法进行性能比较。

总之，openai gym 是一个 RL 算法的测试床（testbed）。

在增强学习中有 2 个基本概念，一个是环境（environment），称为外部世界，另一个为智能体 agent（写的算法）。agent 发送 action 至 environment，environment 返回观察和回报。

gym 的核心接口是 Env，作为统一的环境接口。Env 包含下面几个核心方法：

1、reset(self):重置环境的状态，返回观察。

2、step(self,action):推进一个时间步长，返回 observation，reward，done，info

3、render(self,mode='human',close=False):重绘环境的一帧。默认模式一般比较友好，如弹出一个窗口。

一、 环境构建

0 linux 环境

Ubuntu 16.04 LTS X64（含 Python 2.7.11+，OpenJDK 64-Bit Server VM ）

需下载：Eclipse for linux 64 +python 插件（pydev，http://pydev.org/updates）

注：可以采用 vmware 安装虚拟机的方式。

1 安装 gym（以 ubuntu16.04 为例，使用 root 用户操作）

安装依赖包：

```
apt-get install -y python-numpy python-dev cmake zlib1g-dev libjpeg-dev xvfb libav-tools xorg-dev python-opengl libboost-all-dev libsdl2-dev swig
```

方式 1：Use git:

```
git clone https://github.com/openai/gym
cd gym

pip install -e . # minimal install
pip install -e .[all]  # full install (this requires cmake and a recent pip version)
```

方式 2：use pip：

```
pip install gym  #minimal install

pip install gym[all]  #full install,  fetch gym as a package.
```

二、 运行一个环境（environment）

在 eclipse pydev 视图建立 python 项目，建立一个 python 模块，输入下列代码：
运行一维一级倒立摆环境：

```
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
```

由动画结果可以看出随机控制算法发散，系统很快失去稳定。

3 观察（Observations）

环境的 step 函数返回我们需要的信息，step 函数返回四个值，

1、observation (object):观察，一个与环境相关的对象描述你观察到的环境。如相机的像素信息，机器人的角速度和角加速度，棋盘游戏中的棋盘状态。

2、reward (float)：回报，之前行为获得的所有回报之和。不同环境的计算方式不一，但目标总是增加自己的总回报。

3、done (boolean)：判断是否到了重新设定(reset)环境的时刻了。done 为 true 说明该 episode 完成。

4、info(dict):用于调试的诊断信息。但是，正式的评价这不允许使用该信息进行学习。

这是一个典型的 agent-environment loop 的实现。每一个时间步长，agent 都选择一个 action，environment 返回一个观察和回报。

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

当 done 为真时，控制失败，此阶段 episode 结束。可以计算每次 episode 的回报就是其坚持的 t+1 时间，坚持的越久回报越大。在上面算法中，agent 的行为选择是随机的，平均回报为 20 左右。

4 空间（spaces）

```
import gym
env = gym.make('CartPole-v0')
print(env.action_space)
#> Discrete(2)，非负整数 0 1
print(env.observation_space)
#> Box(4,) 4 维的 box

print(env.observation_space.high)
#> array([ 2.4       ,        inf,  0.41887902,        inf])
print(env.observation_space.low)
#> array([-2.4       ,       -inf, -0.41887902,       -inf])

from gym import spaces
space = spaces.Discrete(8) # Set with 8 elements {0, 1, 2, ..., 7}
x = space.sample()
assert space.contains(x)
assert space.n == 8
```

3.5gym 包含的环境 environments

```
from gym import envs
print(envs.registry.all())
```

3.6 记录和上传结果-使用环境中的 monitor

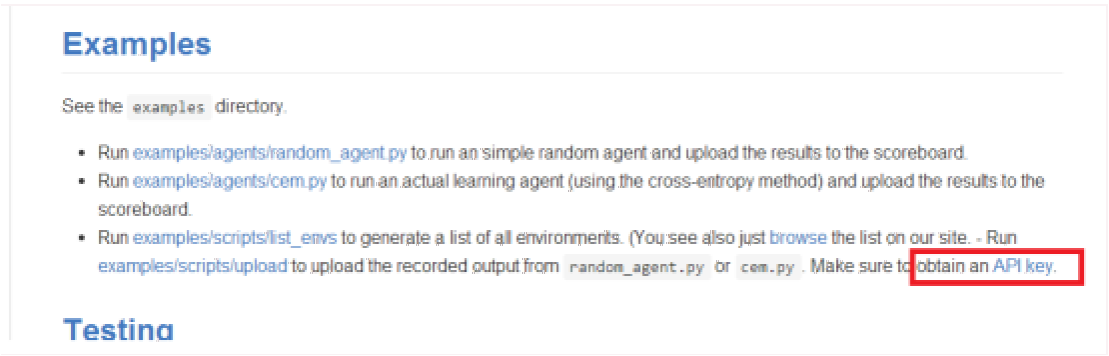
记录：

```
import gym
env = gym.make('CartPole-v0')
env.monitor.start('/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.monitor.close()
```

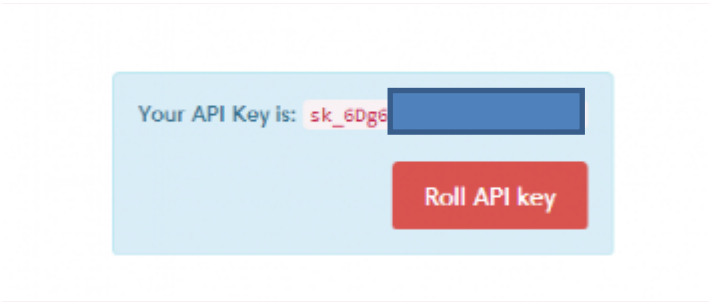
上传：

```
import gym
gym.upload('/tmp/cartpole-experiment-1', api_key='YOUR_API_KEY')
```

获取 API KEY



登录 github 获取 api key

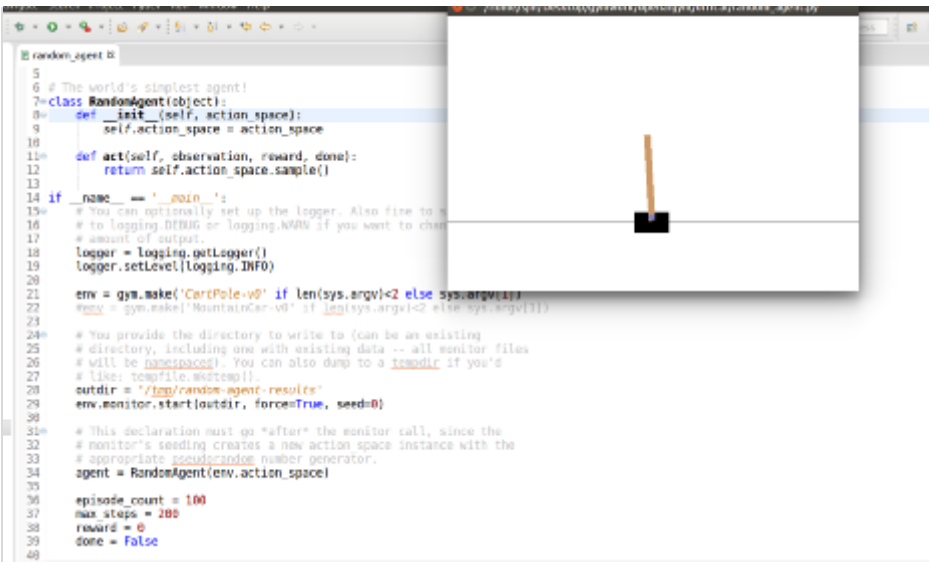


2 运行最简单的 [random agent]

1 random agent ， eg. CartPole-v0, MountainCar-v0

```
import logging
import os, sys

import gym

# The world's simplest agent!
class RandomAgent(object):
    def __init__(self, action_space):
        self.action_space = action_space

    def act(self, observation, reward, done):
        return self.action_space.sample()

if __name__ == '__main__':
    # You can optionally set up the logger. Also fine to set the level
    # to logging.DEBUG or logging.WARN if you want to change the
    # amount of output.
    logger = logging.getLogger()
```

```
    logger.setLevel(logging.INFO)

    env = gym.make('CartPole-v0' if len(sys.argv)<2 else sys.argv[1])

    # You provide the directory to write to (can be an existing
    # directory, including one with existing data -- all monitor files
    # will be namespaced). You can also dump to a tempdir if you'd
    # like: tempfile.mkdtemp().
    outdir = '/tmp/random-agent-results'
    env.monitor.start(outdir, force=True, seed=0)

    # This declaration must go *after* the monitor call, since the
    # monitor's seeding creates a new action_space instance with the
    # appropriate pseudorandom number generator.
    agent = RandomAgent(env.action_space)

    episode_count = 100
    max_steps = 200
    reward = 0
    done = False

    for i in range(episode_count):
        ob = env.reset()

        for j in range(max_steps):
            action = agent.act(ob, reward, done)
            ob, reward, done, _ = env.step(action)
            if done:
                break
            # Note there's no env.render() here. But the environment still can open window and
            # render if asked by env.monitor: it calls env.render('rgb_array') to record video.
            # Video is not recorded every episode, see capped_cubic_video_schedule for details.

    # Dump result info to disk
    env.monitor.close()

    # Upload to the scoreboard. We could also do this from another
    # process if we wanted.
    logger.info("Successfully ran RandomAgent. Now trying to upload results to the scoreboard. If it breaks, you can always just try re-uploading the same results.")
    #gym.upload(outdir)
```



```
# The world's simplest agent!
class RandomAgent(object):
    def __init__(self, action_space):
        self.action_space = action_space

    def act(self, observation, reward, done):
        return self.action_space.sample()

if __name__ == '__main__':
    # You can optionally set up the logger. Also fine to s
    # to logging.DEBUG or logging.WARN if you want to chan
    # amount of output.
    logger = logging.getLogger()
    logger.setLevel(logging.INFO)

    env = gym.make('CartPole-v0' if len(sys.argv)<2 else sys.argv[1])
    #env = gym.make('MountainCar-v0' if len(sys.argv)<2 else sys.argv[1])

    # You provide the directory to write to (can be an existing
    # directory, including one with existing data -- all monitor files
    # will be namespaced). You can also dump to a tempdir if you'd
    # like: tempfile.mkdtemp().
    outdir = '/tmp/random-agent-results'
    env.monitor.start(outdir, force=True, seed=0)

    # This declaration must go *after* the monitor call, since the
    # monitor's seeding creates a new action_space instance with the
    # appropriate pseudo-random number generator.
    agent = RandomAgent(env.action_space)

    episode_count = 100
    max_steps = 200
    reward = 0
    done = False
```

Have fun !