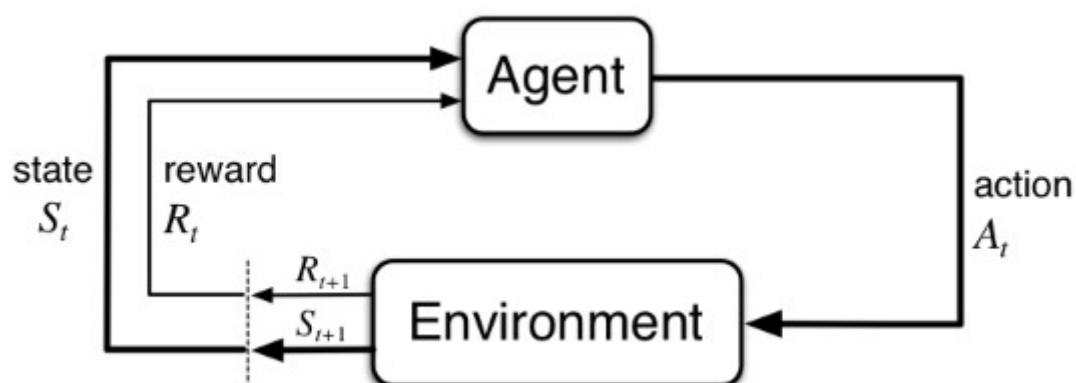


2016 年 5 月 4 日, OpenAI 发布了人工**智能**研究工具集 OpenAI Gym。OpenAI Gym 是一款用于研发和比较学习**算法**的工具包。它与很多数值计算库兼容, 比如 tensorflow 和 theano。现在支持的语言主要是 **Python**。

openai gym 是一个增强学习 (reinforcement learning, RL) 算法的**测试床** (testbed)。增强学习和有监督学习的评测不一样。有监督学习的评测工具是数据。只要提供一批有标注的数据 18:34:13 就能进行有监督学习的评测。增强学习的评测工具是环境。需要提供一个环境给 Agent 运行, 才能评测 Agent 的策略的优劣。OpenAI Gym 是提供各种环境的开源工具包。

增强学习有几个基本概念:

- (1) agent: 智能体, 也就是**机器人**, 你的代码本身。
- (2) environment: 环境, 也就是游戏本身, openai gym 提供了多款游戏, 也就是提供了多个环境。
- (3) action: 行动, 比如玩超级玛丽, 向上向下等动作。
- (4) state: 状态, 每次智能体做出行动, 环境会相应地做出反应, 返回一个状态和奖励。
- (5) reward: 奖励: 根据游戏规则的得分。智能体不知道怎样才能得分, 它通过不断地尝试来理解游戏规则, 比如它在这个状态做出向上的动作, 得分, 那么下一次它处于这个环境状态, 就倾向于做出向上的动作。



OpenAI Gym 由两部分组成:

1. gym 开源库：测试问题的集合。当你测试增强学习的时候，测试问题就是环境，比如机器人玩游戏，环境的集合就是游戏的画面。这些环境有一个公共的接口，允许用户设计通用的算法。
2. OpenAI Gym 服务。提供一个站点（比如对于游戏 cartpole-v0：<https://gym.openai.com/envs/CartPole-v0>）和 api，允许用户对他们的测试结果进行比较。

gym 的代码在这上面：<https://github.com/openai/gym>

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。

安装

1. Linux (没试过):

```
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
```

2. Windows (有两种方法):

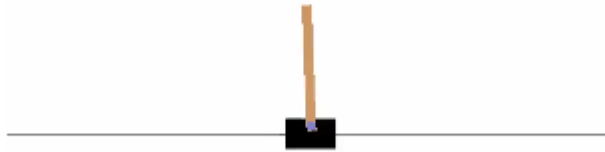
- (1) 使用 pip:

```
pip install gym
```

- (2) 使用 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)
```

接下来以 cartpole-v0 (<https://gym.openai.com/envs/CartPole-v0>) 举例。



这个游戏的规则是让杆不倒。Openai gym 提供了行动的集合，环境的集合等等。Cartpole-v0 来说，动作空间包括向左拉和向右拉两个动作。其实你并不需要关心它的动作空间是什么，当你的学习算法越好，你就越不需要解释这些动作。

运行环境

运行 CartPole-v0 环境 1000 个时间步 (timestep)。

```
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
```

可以看到随机控制算法发散，游戏很快结束。

观察

如果我们想做得好一点，观察周围的环境是必须的。环境的 step 函数返回四个值：

- Observation(object): 返回一个特定环境的对象，描述对环境的观察。比如，来自相机的像素数据，机器人的关节角度和关节速度，或棋盘游戏中的棋盘状态。
- Reward(float): 返回之前动作收获的总的奖励值。不同的环境计算方式不一样，但总体的目标是增加总奖励。
- Done(boolean): 返回是否应该重新设置 (reset) 环境。大多数游戏任务分为多个环节 (episode)，当 done=true 的时候，表示这个环节结束了。
- Info(dict): 用于调试的诊断信息（一般没用）。

这是一个典型的“智能体-环境循环”的实现。每个时间步长 (timestep)，智能体选择一个行动，环境返回一个观察和奖励值。

过程一开始调用 `reset`, 返回一个初始的观察。并根据 `done` 判断是否再次 `reset`。

```
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
```

以上代码有 20 个 episode, 打印每次的环境观察值, 随机采取行动, 返回环境的观察值、奖励、done 和调试信息。当 done 为 true 时, 该 episode 结束, 开始下一个 episode。可以看到, 观察了环境, 每次坚持的时间好像稍微长了一点。

运行结果如下:

```
[-0.061586  -0.75893141  0.05793238  1.15547541]
[-0.07676463 -0.95475889  0.08104189  1.46574644]
[-0.0958598  -1.15077434  0.11035682  1.78260485]
[-0.11887529 -0.95705275  0.14600892  1.5261692 ]
[-0.13801635 -0.7639636  0.1765323  1.28239155]
[-0.15329562 -0.57147373  0.20218013  1.04977545]
Episode finished after 14 timesteps
[-0.02786724  0.00361763 -0.03938967 -0.01611184]
[-0.02779488 -0.19091794 -0.03971191  0.26388759]
[-0.03161324  0.00474768 -0.03443415 -0.04105167]
```

空间

以上, 我们都是从环境的动作空间中随机选择动作。每个环境都有一个 `space` 对象, 用来描述有效的动作和观察:

```
import gym
env = gym.make('CartPole-v0')
print(env.action_space)
#> Discrete(2)
print(env.observation_space)
#> Box(4,)
```

在上面这个例子中，action 取非负整数 0 或 1。Box 表示一个 n 维的盒子，因此 observation 是一个 4 维的数组。我们可以试试 box 的上下限。

```
print(env.observation_space.high)
#> array([ 2.4          ,          inf,  0.20943951,          inf])
print(env.observation_space.low)
#> array([-2.4          ,          -inf, -0.20943951,          -inf])
```

Box 和 discrete 是最常见的 space。你可以从 space 中取样或者验证是否属于它。

```
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
```

环境

Gym 的主要目的是提供一个大环境集合，具有一个公共接口，并且允许比较算法。你可以列出这些环境。

```
from gym import envs
print(envs.registry.all())
#> [EnvSpec(DoubleDunk-v0), EnvSpec(InvertedDoublePendulum-v0),
EnvSpec(BeamRider-v0), EnvSpec(Phoenix-ram-v0), EnvSpec(Asterix-v0),
EnvSpec(TimePilot-v0), EnvSpec(Alien-v0), EnvSpec(Robotank-ram-v0),
EnvSpec(CartPole-v0), EnvSpec(Berzerk-v0), EnvSpec(Berzerk-ram-v0),
EnvSpec(Gopher-ram-v0), ...]
```

这列出了一系列的 EnvSpec。它们为特定任务定义特定参数，包括运行的实验数目和最多的步数。比如，[EnvSpec\(Hopper-v1\)](#) 定义了一个环境，环境的目标是让一个 2D 的模拟机器跳跃。[EnvSpec\(Go9x9-v0\)](#) 定义了 9*9 棋盘上的围棋游戏。

这些环境 ID 被视为不透明字符串。为了确保与未来的有效比较，环境永远不会以影响性能的方式更改，只能由较新的版本替代。我们目前使用 v0 为每个环境添加后缀，以便将来的替换可以自然地称为 v1, v2 等。

记录和加载结果

Gym 使得记录算法在环境中的性能变得简单，同时能记录学习过程的视频。只要使用 monitor 如下：

```
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()

```

跑了一下发现有错：

env.monitor is deprecated. Wrap your env with gym.wrappers.Monitor to record data.

改为如下：

```

from gym.wrappers import Monitor
env =
Monitor(directory='/tmp/cartpole-experiment-0201', video_callable=False,
write_upon_reset=True)(env)
env.close()

```

(mark 一下找 bug 思路，gym\monitoring\tests 里面有测试的案例，参考 test_monitor.py 写代码。)

产生的结果放到'/tmp/cartpole-experiment-1'这个文件夹中，你可以加载到评分板。

Finished writing results. You can upload them to the scoreboard via gym.upload('E:\\tmp\\cartpol

monitor 支持将一个环境的多个案例写入一个单独的目录。

然后你可以把你的结果加载到 OpenAI Gym：

```

import gym
gym.upload('/tmp/cartpole-experiment-1', api_key='
sk_FYp0GcldQU69epifs7ZE6w')

```

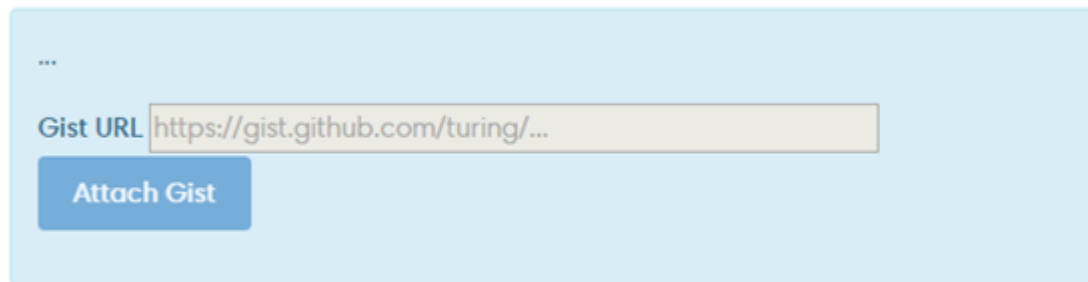
输出应该是这样：

```
[2016-04-22 23:16:03,123] Uploading 20 episodes of training data
[2016-04-22 23:16:04,194] Uploading videos of 2 training episodes (6306
bytes)
[2016-04-22 23:16:04,437] Creating evaluation object on the server with
learning curve and training video
[2016-04-22 23:16:04,677]
*****
You successfully uploaded your agent evaluation to
OpenAI Gym! You can find it at:
```

https://gym.openai.com/evaluations/eval_tmX7tssiRVtYzZkOP1WhKA

估值

每次加载会产生一个估值对象。官方文件说，应该创建一个 Gist（被墙了）显示怎么复制你的结果。估值页会有一个如下方框：



The screenshot shows a light blue rectangular box. Inside, at the top left, are three dots "...". Below them, the text "Gist URL" is followed by a text input field containing the URL "https://gist.github.com/turing/...". Below the input field is a blue button with the white text "Attach Gist".

你还可以在加载的时候提供你的 gist，通过一个 `writeup` 参数

```
import gym
gym.upload('/tmp/cartpole-experiment-1',
writeup='https://gist.github.com/gdb/b6365e79be6052e7531e7ba6ea8caf23',
api_key='sk_FYp0GcldQU69epifs7ZE6w')
```

这一步是将你的结果提交到在线网站上进行评估，你的结果会被自动评分，并且会产生一个漂亮的界面，如：

https://gym.openai.com/evaluations/eval_Ir5NHkdNRGqympBDcdNNNw

在大多数环境中，你的目标是用最少的步数达到性能的要求（有一个阈值）。而在一些特别复杂的环境中，阈值是什么还不清楚，因此，你的目标是最优化性能。

注意，现在 writeup 被舍弃了，所以不需要 writeup。

另外，api_key 是指

再 mark 一个错误:

```
raise error.Error("%s] You didn't have any recorded training data in {}.
Once you've used 'env.monitor.start(training_dir)' to start recording,
you need to actually run some rollouts. Please join the community chat
on https://gym.openai.com if you have any issues.".format(env_id,
training_dir))
```

这是由于之前更改 `monitor`，改成调用 `wrappers` 里面的文件时候出的错。将结果放到 `/tmp/cartpole-experiment-1` 这个文件夹的时候出错，也就是在 `monitor` 这一步出错，通过查看文件夹里面的文件也可看到文件夹里面的文件大小都很小，内容也不对，如下：

| 名称 | 类型 | 大小 | 修改日期 |
|--|-----------|------|------------------|
| openaigym.episode_batch.0.20092.stats.json | JSON File | 1 KB | 2016/12/27 16:51 |
| openaigym.manifest.0.20092.manifest.json | JSON File | 1 KB | 2016/12/27 16:51 |

```
openai.gym.episode_batch.0.20092.stats.json
```

因此从 monitor 入手更改。

改完之后产生的文件如下:

```

{"initial_reset_timestamp": 1482825342.885998, "episode_types": ["t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t", "t",
"t"], "timestamps": [1482825343.5884657], "episode_lengths": [200],
"episode_rewards": [200.0]}

```