强化学习基础篇（九）OpenAI Gym基础介绍

1. Gym介绍

Gym是一个研究和开发强化学习相关算法的仿真平台，无需智能体先验知识，由以下两部分组成

- Gym开源库：测试问题的集合。当你测试强化学习的时候，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。
- OpenAI Gym服务：提供一个站点和API（比如经典控制问题：CartPole-v0），允许用户对他们的测试结果进行比较。

2. Gym安装

我们需要在Python 3.5+的环境中简单得使用pip安装gym

```
pip install gym
```

如果需要从源码安装gym，那么可以：

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

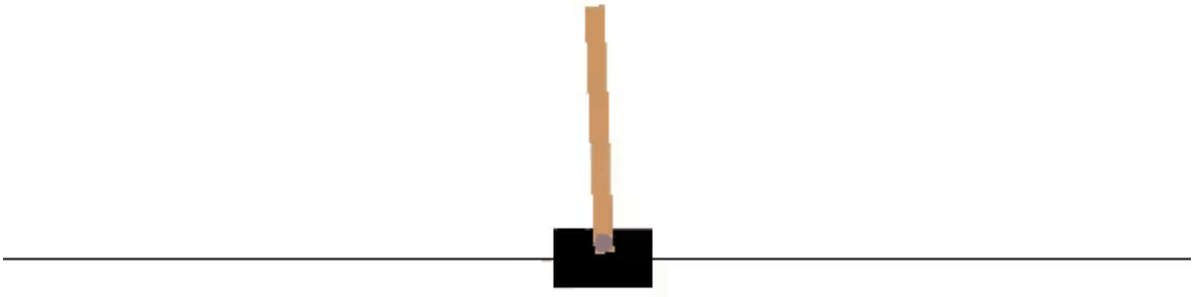
可以运行pip install -e .[all]执行包含所有环境的完整安装。 这需要安装一些依赖包，包括cmake和最新的pip版本。

3. Gym使用demo

简单来说OpenAI Gym提供了许多问题和环境（或游戏）的接口，而用户无需过多了解游戏的内部实现，通过简单地调用就可以用来测试和仿真。接下来以经典控制问题CartPole-v0为例，简单了解一下Gym的特点

```
# 导入gym环境
import gym
# 声明所使用的环境
env = gym.make('CartPole-v0')
# 环境初始化
env.reset()

# 对环境进行迭代执行1000次
for _ in range(1000):
    env.render()
    observation, reward, done, info = env.step(env.action_space.sample()) # 采取随机动作
    if done:
       env.reset()
env.close()
```

运行效果如下



以上代码中可以看出，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：

- reset(self)：重置环境的状态，返回观察。
- step(self, action)：推进一个时间步长，返回observation, reward, done, info。
- 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。
- close(self)：关闭环境，并清除内存

以上代码首先导入*gym*库，然后创建*CartPole-v0*环境，并重置环境状态。在for循环中进行*1000*个时间步长控制，env.render()刷新每个时间步长环境画面，对当前环境状态采取一个随机动作（0或1），在环境返回done为True时，重置环境，最后循环结束后关闭仿真环境。

4、观测（Observations)

在上面代码中使用了*env.step()*函数来对每一步进行仿真，在*Gym*中，*env.step()*会返回 4 个参数：

- 观测 *Observation (Object)*：当前*step*执行后，环境的观测(类型为对象)。例如，从相机获取的像素点，机器人各个关节的角度或棋盘游戏当前的状态等；
- 奖励 *Reward (Float)*: 执行上一步动作(*action*)后，智能体(*agent*)获得的奖励(浮点类型)，不同的环境中奖励值变化范围也不相同，但是强化学习的目标就是使得总奖励值最大；
- 完成 *Done (Boolen)*: 表示是否需要将环境重置 *env.reset*。大多数情况下，当 *Done* 为*True* 时，就表明当前回合(*episode*)或者试验(*tial*)结束。例如当机器人摔倒或者掉出台面，就应当终止当前回合进行重置(*reset*);
- 信息 *Info (Dict)*: 针对调试过程的诊断信息。在标准的智体仿真评估当中不会使用到这个*info*，具体用到的时候再说。

在 *Gym* 仿真中，每一次回合开始，需要先执行 *reset()* 函数，返回初始观测信息，然后根据标志位 *done* 的状态，来决定是否进行下一次回合。所以更恰当的方法是遵守*done*的标志。

```python
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
env.close()
```

代码运行结果的片段如下所示：

```
[ 0.04025062 -0.04312649  0.00186348  0.02288173]
[ 0.03938809 -0.23827512  0.00232111  0.31615203]
[ 0.03462259 -0.43343005  0.00864416  0.60956605]
[ 0.02595398 -0.23843     0.02083548  0.31961824]
[ 0.02118538 -0.43384239  0.02722784  0.6187984 ]
[ 0.01250854 -0.23911113  0.03960381  0.33481376]
[ 0.00772631 -0.43477369  0.04630008  0.63971794]
[-0.00096916 -0.63050954  0.05909444  0.94661444]
[-0.01357935 -0.43623107  0.07802673  0.67306909]
[-0.02230397 -0.24227538  0.09148811  0.40593731]
[-0.02714948 -0.43856752  0.09960686  0.72600415]
[-0.03592083 -0.24495361  0.11412694  0.46625881]
[-0.0408199  -0.05161354  0.12345212  0.21161588]
[-0.04185217  0.14154693  0.12768444 -0.03971694]
[-0.03902123 -0.05515279  0.1268901   0.29036807]
[-0.04012429 -0.25183418  0.13269746  0.6202239 ]
[-0.04516097 -0.05879065  0.14510194  0.37210296]
[-0.04633679  0.13400401  0.152544    0.12846047]
[-0.04365671 -0.06293669  0.15511321  0.46511532]
[-0.04491544 -0.25987115  0.16441551  0.80239106]
[-0.05011286 -0.45681992  0.18046333  1.14195086]
[-0.05924926 -0.65378152  0.20330235  1.48536419]
Episode finished after 22 timesteps
```

上面的结果可以看到这个迭代中，输出的观测为一个列表。这是CartPole环境特有的状态，其规则是$[x, \theta, \dot{d}, \dot{\theta}]$。

其中：

- x表示小车在轨道上的位置（*position of the cart on the track*）
- θ表示杆子与竖直方向的夹角（*angle of the pole with the vertical*）
- $\dot{x}$表示小车速度（*cart velocity*）
- $\dot{\theta}$表示角度变化率（*rate of change of the angle*）

5、空间（*Spaces*）

每次执行的动作(*action*)都是从环境动作空间中随机进行选取的，但是这些动作 (*action*) 是什么?在 *Gym* 的仿真环境中，有运动空间 *action_space* 和观测空间*observation_space* 两个指标，程序中被定义为 *Space*类型，用于描述有效的运动和观测的格式和范围。下面是一个代码示例：

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

```
Discrete(2)
Box(-3.4028234663852886e+38, 3.4028234663852886e+38, (4,), float32)
```

从程序运行结果可以看出：

- action_space 是一个离散Discrete类型，从discrete.py源码可知，范围是一个{0,1,...,n-1} 长度为 n 的非负整数集合，在CartPole-v0例子中，动作空间表示为{0,1}。
- observation_space 是一个Box类型，从box.py源码可知，表示一个 n 维的盒子，所以在上一节打印出来的observation是一个长度为 4 的数组。数组中的每个元素都具有上下界。

6. 奖励(reward)

在*gym*的*Cart Pole*环境（*env*）里面，左移或者右移小车的*action*之后，*env*会返回一个+1的*reward*。其中*CartPole-v0*中到达200个*reward*之后，游戏也会结束，而*CartPole-v1*中则为*500*。最大奖励（*reward*）阈值可通过前面介绍的注册表进行修改。

7. 注册表

*Gym*是一个包含各种各样强化学习仿真环境的大集合，并且封装成通用的接口暴露给用户，查看所有环境的代码如下

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

8.注册模拟器

*Gym*支持将用户制作的环境写入到注册表中，需要执行 *gym.make()*和在启动时注册register。如果要注册自己的环境，那么假设你在以下结构中定义了自己的环境：

```
myenv/
    __init__.py
    myenv.py
```

i. myenv.py`包含适用于我们自己的环境的类。 在`init.py中，输入以下代码：

```
from gym.envs.registration import register
register(
    id='MyEnv-v0',
    entry_point='myenv.myenv:MyEnv', # 第一个myenv是文件夹名字，第二个myenv是文件名字，MyEnv是文件内类的名字
)
```

ii. 要使用我们自己的环境：

```
import gym
import myenv # 一定记得导入自己的环境，这是很容易忽略的一点
env = gym.make('MyEnv-v0')
```

iii. 在PYTHONPATH中安装`myenv`目录或从父目录启动python。

```
目录结构：
myenv/
    __init__.py
    my_hotter_colder.py
-------------------
__init__.py 文件：
-------------------
from gym.envs.registration import register
register(
    id='MyHotterColder-v0',
    entry_point='myenv.my_hotter_colder:MyHotterColder',
)
-------------------
my_hotter_colder.py文件：
-------------------
import gym
from gym import spaces
from gym.utils import seeding
import numpy as np

class MyHotterColder(gym.Env):
    """Hotter Colder
    The goal of hotter colder is to guess closer to a randomly selected
number

    After each step the agent receives an observation of:
    0 - No guess yet submitted (only after reset)
    1 - Guess is lower than the target
    2 - Guess is equal to the target
    3 - Guess is higher than the target

    The rewards is calculated as:
    (min(action, self.number) + self.range) / (max(action, self.number) +
self.range)

    Ideally an agent will be able to recognise the 'scent' of a higher
reward and
    increase the rate in which is guesses in that direction until the reward
reaches
    its maximum
    """
    def __init__(self):
        self.range = 1000  # +/- value the randomly select number can be
between
        self.bounds = 2000  # Action space bounds

        self.action_space = spaces.Box(low=np.array([-self.bounds]),
high=np.array([self.bounds]))
        self.observation_space = spaces.Discrete(4)

        self.number = 0
        self.guess_count = 0
        self.guess_max = 200
        self.observation = 0

```

```
        self.seed()
        self.reset()

    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]

    def step(self, action):
        assert self.action_space.contains(action)

        if action < self.number:
            self.observation = 1

        elif action == self.number:
            self.observation = 2

        elif action > self.number:
            self.observation = 3

        reward = ((min(action, self.number) + self.bounds) / (max(action, self.number) + self.bounds)) ** 2

        self.guess_count += 1
        done = self.guess_count >= self.guess_max

        return self.observation, reward[0], done, {"number": self.number, "guesses": self.guess_count}

    def reset(self):
        self.number = self.np_random.uniform(-self.range, self.range)
        self.guess_count = 0
        self.observation = 0
        return self.observation
```

9. OpenAI Gym评估平台

用户可以记录和上传算法在环境中的表现或者上传自己模型的*Gist*，生成评估报告，还能录制模型玩游戏的小视频。在每个环境下都有一个排行榜，用来比较大家的模型表现。

上传于录制方法如下所示

```
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
```

使用Monitor Wrapper包装自己的环境，在自己定义的路径下将记录自己模型的性能。支持将一个环境下的不同模型性能写在同一个路径下。

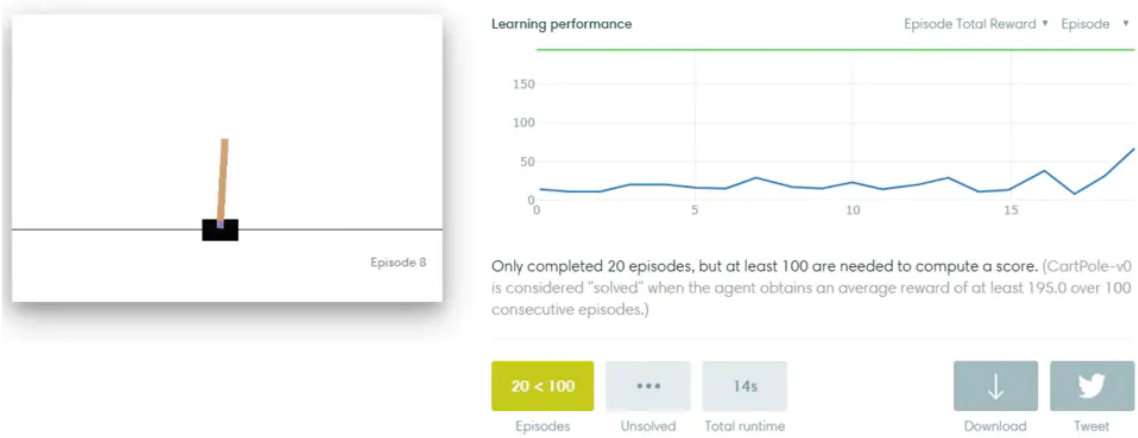
在官网注册账号后，可以在个人页面上看到自己的API_Key，接下来可以将结果上传至OpenAI Gym：

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

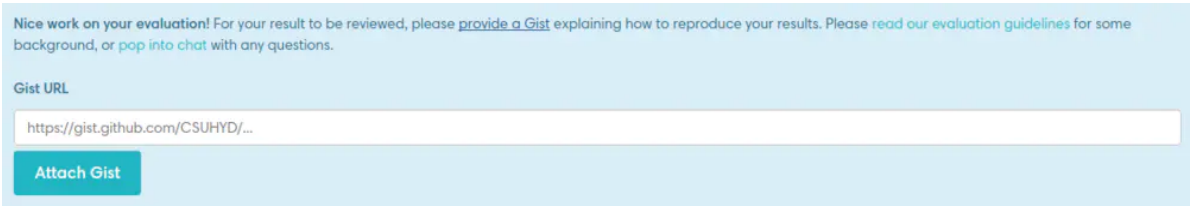
然后得到如下结果：

```
***********************************************************
You successfully uploaded your evaluation on CartPole-v0 to
OpenAI Gym! You can find it at:

    https://gym.openai.com/evaluations/eval_hxhpHLIqRaGT2xyN3KaAoQ

***********************************************************
```

打开链接会有当前模型在环境下的评估报告，并且还录制了小视频：



每次上传结果，OpenAI Gym都会对其进行评估。



创建一个Github Gist将结果上传，或者直接在upload时传入参数：

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

评估将自动计算得分，并生成一个漂亮的页面。

在大多数环境中，我们的目标是尽量减少达到阈值级别的性能所需的步骤数。不同的环境都有不同的阈值，在某些环境下，尚不清楚该阈值是什么，此时目标是使最终的表现最大化。在cartpole这个环境中，阈值就是立杆能够直立的帧数。