

# Gym 强化学习入门到提高



@神奇的战士-王松

<https://github.com/wangshub>

# 学习目的

- 了解强化学习基本概念
- 利用 Python 进行强化学习仿真
- 强化学习快速入门
  
- 学习基础
  - Python 语法基础
  - 基本矩阵运算知识
  - 基本的概率统计知识

# 内容简介

- 强化学习基本理论
- 强化学习仿真环境 Gym
- 第一个 Hello world
- 求解 Cartpole 模型
- 学习进阶与总结

# 介绍

## ➤ 强化学习(Reinforcement learning, RL)

- 机器学习的一个子领域,用于制定决策和运动自由度控制
- 是指一类与环境交互中不断学习的问题以及解决这类问题的方法

## ➤ 优点

- 强化学习非常通用,可以用来解决需要作出一些列决策的所有问题
- 强化学习已经可以在许多复杂的环境中取得较好的实验结果



# 监督学习与强化学习

- 有监督学习
  - 从有标记的数据中推导函数关系，给定数据，预测标签；
  - 例如：花草识别
- 无监督学习
  - 从无标签的数据中推断结论，给定数据，寻找隐藏的函数结构；
  - 例如：聚类
- 强化学习
  - 智体在环境中行动获得最大累积奖励，学习一些列动作，获得最大收益；
  - 例如：AlphaGo 下棋

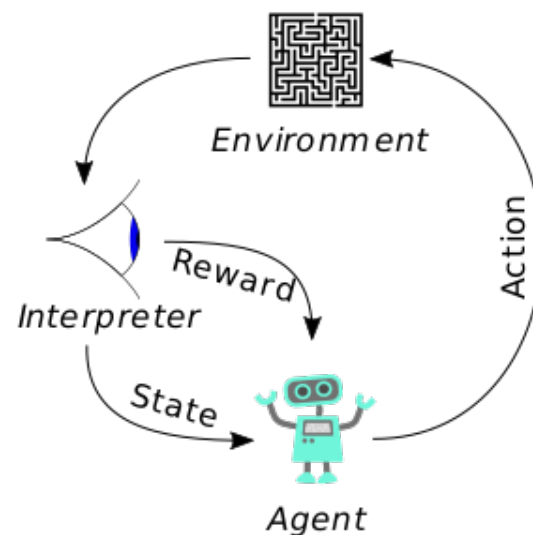
# 强化学习的基本概念

- 强化学习和监督学习的不同在于，强化学习问题不需要给出“正确”策略作为监督信息，只需要给出策略的(延迟)回报，并通过调整策略来取得最大化的期望回报。
- 对于环境反馈有利的奖励，智体将会强化引发这种奖励的动作，并且在之后的环境交互过程中更加偏向于这种行为和动作。
- 本质：奖惩和试错(Trial and Error)



# 强化学习的术语

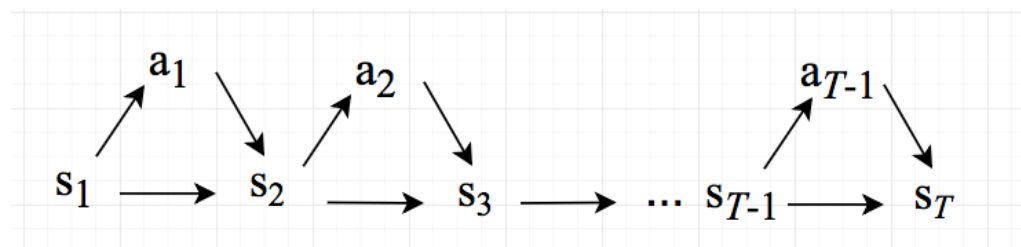
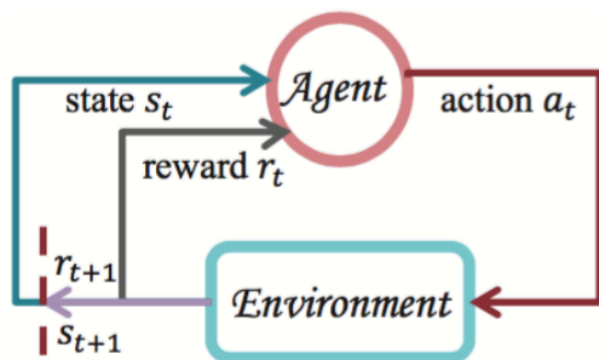
- 智体(agent)
- 环境(Environment)
- 状态(State)  $s$
- 动作(Action)  $a$
- 策略(Policy)  $\pi$
- 奖励(Reward)  $r$
- 状态转移概率



# 问题建模

## ➤ 马尔科夫决策过程(MDP)

- 状态空间  $S$ : Agent 可能感知的所有状态
- 动作空间  $A$ : Agent 在每个状态可以采取的所有动作
- 奖励函数  $R$ : 在状态  $s$  上执行动作  $a$ , 状态转移到  $s'$  获得的奖励
- 状态转移函数  $T$ : 环境从  $s$  转移到  $s'$  的概率



$$\tau = s_0, a_0, s_1, r_1, a_1, \dots, s_{T-1}, s_T, r_T$$



# MDP 模型

## ➤ 目标

- 返回的奖励期望最大

## ➤ 策略

- 状态到大屏幕和自拍的映射

## ➤ 最佳策略

- 从任何一个状态开始，均是最优的策略

## ➤ 定理：

- 肯定存在一个确定性的最优策略

# 优化参数

- 折扣率  $\gamma \in 0, 1$

$$G_T = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$$

- 最优策略定义

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right\}, \forall s \in S, \forall t \geq 0.$$

- 寻找最优的状态值函数

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right\}, \forall s \in S, \forall t \geq 0$$

- 最优状态动作值函数

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right\}, \forall s \in S, \forall a \in A, \forall t \geq 0$$

# MDP小结

状态集合,  $|S|=n$ .  $s \in S$

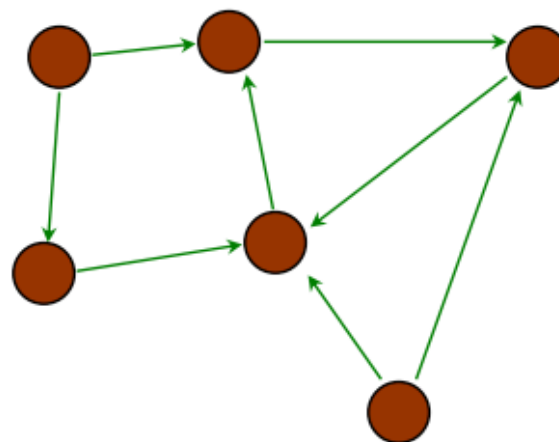
动作集合,  $|A|=k$ .  $a \in A$

转移函数  $\delta(s_1, a, s_2)$

即时奖赏函数  $R(s, a)$

策略  $\pi: S \rightarrow A$

折扣累计返回  $\sum_{i=0}^{\infty} \gamma^i r_i$

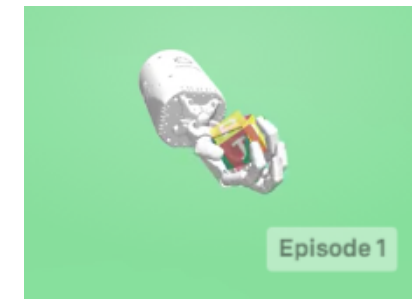
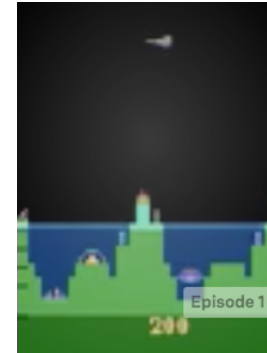
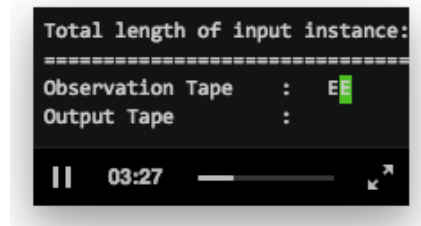


# 更多强化学习内容

- 《Reinforcement Learning: An Introduction》
  - <http://incompleteideas.net/book/the-book-2nd.html>
- 《Reinforcement Learning》
  - <https://cn.udacity.com/course/reinforcement-learning--ud600>

# OpenAI Gym

- Algorithm
- Atari
- Box2D
- Classic control
- MuJoCo
- Robotics
- Toy text
- Roboschool
- Pybullet



# Gym 仿真环境

## ➤ 环境安装

- pip 安装
  - `pip3 install gym`
- 源码安装
  - `git clone https://github.com/openai/gym.git`  
`cd gym`  
`pip install -e .`

# 验证环境是否安装正确

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

- Total length of input instance: 3, step: 0

=====

Observation Tape : [42mD[0mBC

Output Tape :

Targets : DBC

<ipykernel.iostream.OutputStream at 0x1043684e0>

# Gym 术语

- 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 (Boolean): 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。



# OpenAI 设计

- 注重环境，而非智体(Agent)；
- 注重样本复杂度，而非最终的性能；
- 注重同行评审，而非是比分；
- 严格的环境版本控制；
- 环境监控；



# OpenAI Episode

```
ob0 = env.reset() # sample environment state, return first observation
```

```
a0 = agent.act(ob0) # agent chooses first action
```

```
ob1, rew0, done0, info0 = env.step(a0) # environment returns observation,  
# reward, and boolean flag indicating if the episode is complete.
```

```
a1 = agent.act(ob1)
```

```
ob2, rew1, done1, info1 = env.step(a1)
```

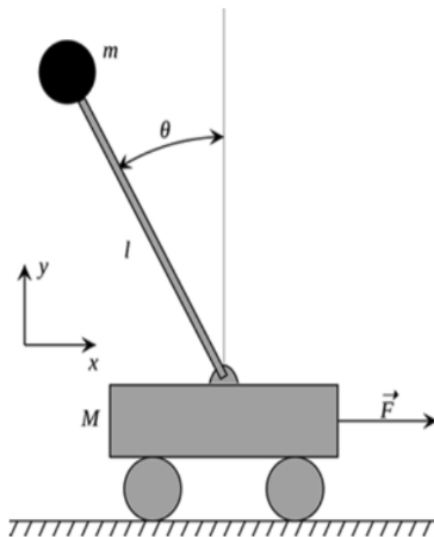
```
...
```

```
a99 = agent.act(o99)
```

```
ob100, rew99, done99, info2 = env.step(a99)
```

```
# done99 == True => terminal
```

# 第一个强化学习 Hello World



Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	$-\text{Inf}$	$\text{Inf}$
2	Pole Angle	$\sim -41.8^\circ$	$\sim 41.8^\circ$
3	Pole Velocity At Tip	$-\text{Inf}$	$\text{Inf}$

Num	Action
0	Push cart to the left
1	Push cart to the right

# 观测与运动空间

## ■ 观测空间

`observation_space` 是一个 `Box` 类型，从 `box.py` 源码可知，表示一个  $n$  维的盒子，`CartPole-v0` 例子中 `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；

## ■ 运动空间

运动空间 `action_space` 是一个离散 `Discrete` 类型，从 `discrete.py` 源码可知，范围是一个  $\{0, 1, \dots, n-1\}$  长度为  $n$  的非负整数集合，在 `CartPole-v0` 例子中，动作空间表示为  $\{0, 1\}$ ；

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

print(env.observation_space.high)
print(env.observation_space.low)
```


```
WARN: gym.spaces.Box autodetected dtype as <class 'numpy.float32'>. Please provide explicit dtype.
Discrete(2)
Box(4,)
[4.80000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.80000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

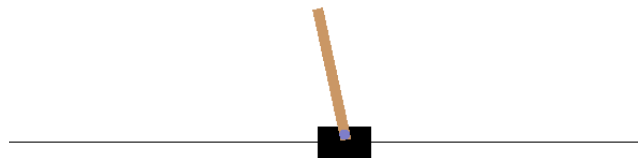
# 一个 Hello World

```
import gym
env = gym.make('CartPole-v0')
init_state = env.reset()
print('init state = ', init_state)
for _ in range(100):
    env.render()
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action) # take a random action
    if done:
        env.render()
        break
```

WARN: gym.spaces.Box autodetected dtype as <class 'numpy.float32'>. Please provide explicit dtype.

init state = [ 0.03108121 0.02741301 -0.01358363 -0.03507543]

 /usr/local/lib/python3.7/site-packages/ipykernel\_launcher.py



# 概念解读

## ➤ 创建实例

- 每个 Gym 环境都有唯一的命名，命名方式为  
([A-Za-z0-9]+-)v([0-9]+)
- 使用 `gym.make('CartPole-v0')` 创建环境

## ➤ 重置函数 `reset`

- 用于重新开启一个新的回合(试验)
- 返回回合的初始状态



## ➤ 执行(step)

- 执行特定的动作，返回状态(state)
- observation, reward, done, info

## ➤ 渲染(render)

- 用于显示当前环境的状态
- 用于调试和定性的分析不同策略的效果

# 回合终止的条件

- 当满足下列条件之一时，当前回合结束
  - 杆的角度超过 12度
  - 以中点为原点，小车位置超过  $\pm 2.4$
  - 回合长度超过 200 次



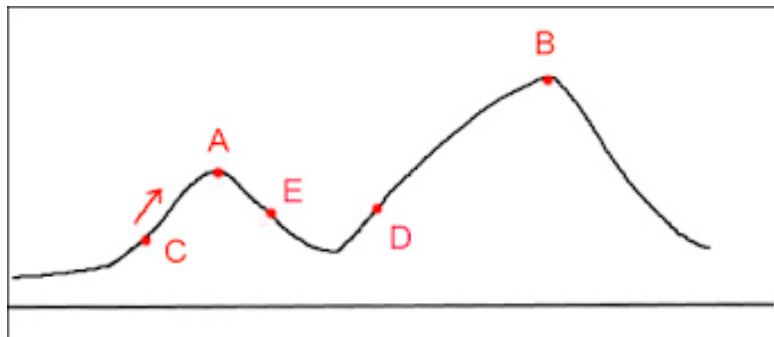
# 求解 cartpole 的一些方法

- 随机猜测算法(The random guessing)
  - 以某种分布产生若干参数或配置，从中选取最佳参数
- 爬山算法(The hill-climbing)
  - 从临近空间选取最优解作为当前解，直到达到局部最优
- 策略梯度算法(Policy gradient)
  - 通过反馈来调整策略，当得到正向反馈时，提高相应动作的概率。

# 简单的爬山算法

## ➤ 算法步骤

- 起始设置为一个随机初始参数；
- 给配置参数添加少量噪声；
- 如果效果优于上一次的参数，则更新；
- 重复上述步骤

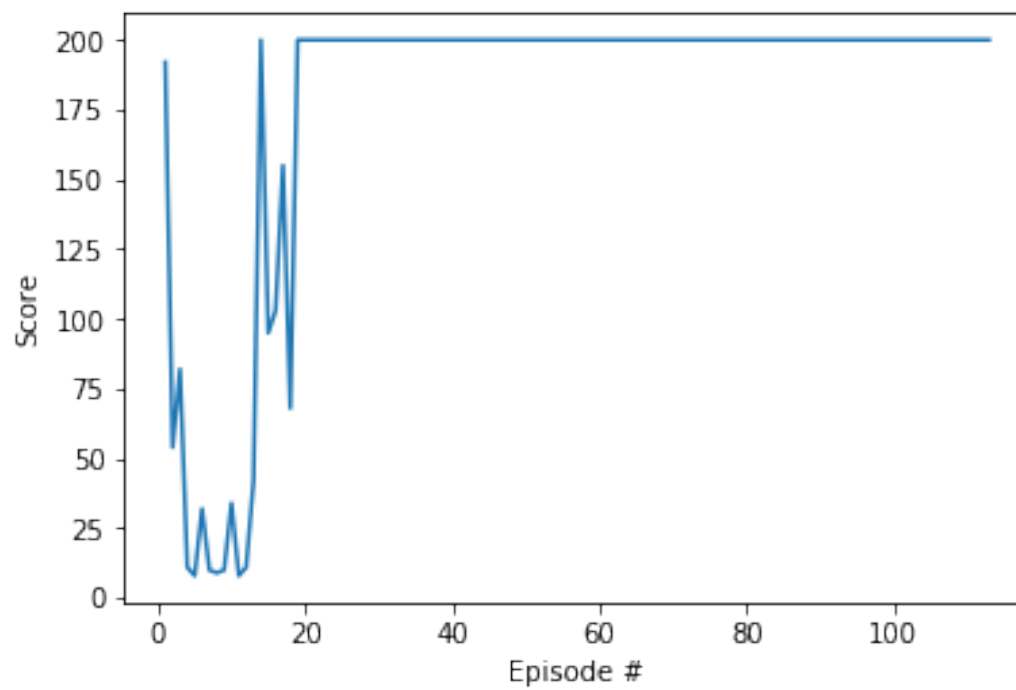


# 代码解读

## ➤ Hill\_Climbing.ipynb

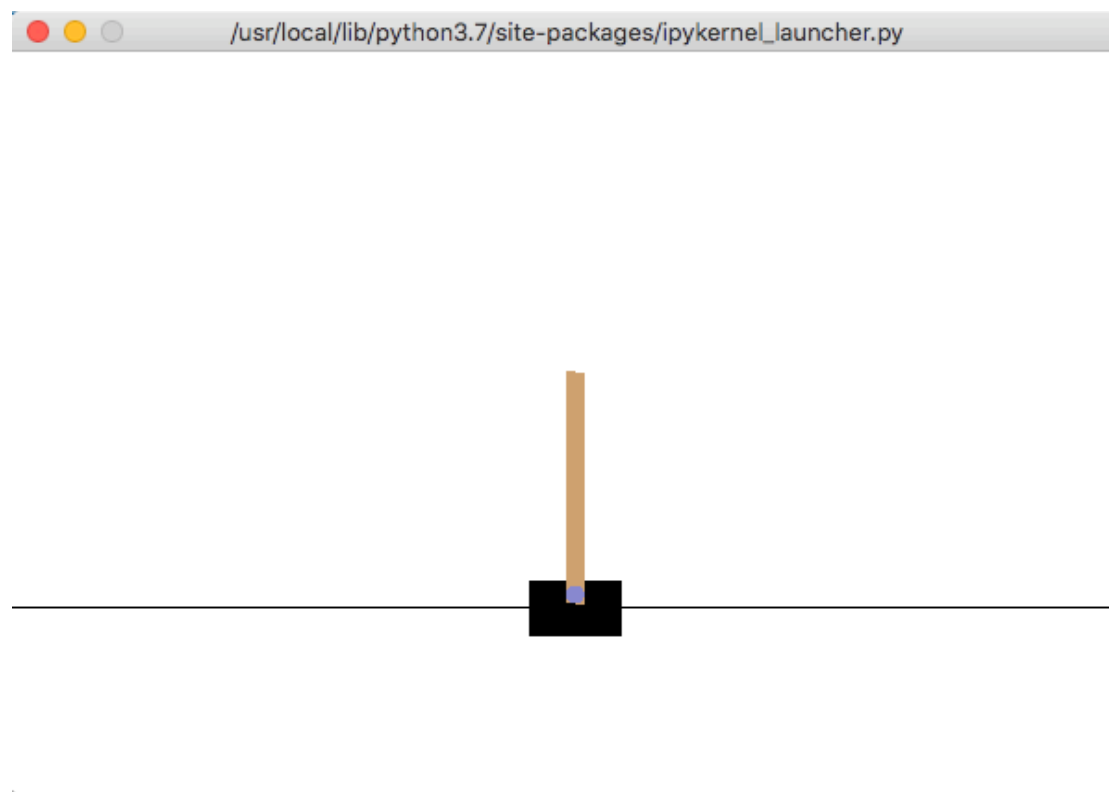
- 导入库
- 定义 Policy
- 随机策略搜索
- Matplot 绘制结果
- 强化学习动画

# 仿真结果



# 仿真结果

- 一个在设定时间内稳定的倒立摆



# 提高阅读

- 今日课件地址 <https://github.com/wangshub>
- OpenAI 官网 <https://openai.com>
- Algorithms for Reinforcement Learning <https://sites.ualberta.ca/~szepesva/RLBook.html>
- Decision Making Under Uncertainty: Theory and Application
- An Introduction to Reinforcement Learning