Hands-On Reinforcement Learning With Python——Temporal Difference Learning

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

更多请看：Reinforcement Learning - 随笔分类 - 凯鲁嘎吉 - 博客园 https://www.cnblogs.com/kailugaji/category/2038931.html

本文代码下载：https://github.com/kailugaji/Hands-on-Reinforcement-Learning/tree/main/01%20Temporal%20Difference%20Learning

$\epsilon $-贪心策略：

$$\pi^{\epsilon}(s)=\begin{cases}\pi(s), & \text{按概率}1-\epsilon,\\ \text{随机选择}\mathcal{A}\text{中的动作}, & \text{按概率}\epsilon.\end{cases} \tag{14.31}$$

1. SARSA

1.1 算法流程

算法 14.3　SARSA：一种同策略的时序差分学习算法

输入: 状态空间 $\mathcal{S}$，动作空间 $\mathcal{A}$，折扣率 γ，学习率 α

$\forall s, \forall a$，随机初始化 $Q(s,a)$；根据 Q 函数构建策略 π;

repeat

　初始化起始状态 s; 选择动作 $a = \pi^{\epsilon}(s)$;　　// $\pi^{\epsilon}(s)$ 参见公式(14.31)

　repeat

　　执行动作 a，得到即时奖励 r 和新状态 s';

　　在状态 s'，选择动作 $a' = \pi^{\epsilon}(s')$;

　　$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$;　　// 更新 Q 函数

　　$\pi(s) = \arg\max_{a \in |\mathcal{A}|} Q(s,a)$;　　// 更新策略

　　$s \leftarrow s', a \leftarrow a'$;

　until s 为终止状态;

until $\forall s, a, Q(s,a)$ 收敛;

输出: 策略 $\pi(s)$

1.2 Python程序

```
# -*- coding: UTF-8 -*-
# Solving the Taxi Problem using SARSA
# From: https://github.com/AndyYue1893/Hands-On-Reinforcement-Learning-With-Python
# https://www.cnblogs.com/kailugaji/ - 凯鲁嘎吉 - 博客园
'''
出租车调度
这里有 4 个地点，分别用 4 个字母表示，任务是要从一个地点接上乘客，送到另外 3 个中的一个放下乘客，越快越好。
颜色：蓝色：乘客，红色：乘客的目的地，黄色：空出租车，绿色：出租车满座，其中 ":" 栅栏可以穿越，"|" 栅栏不能穿越
Reward：成功运送一个客人获得 20 分奖励
        每走一步损失 1 分（希望尽快送到目的地）
        没有把客人放到指定的位置，损失 10 分
Action: 0：向南移动，1：向北移动，2：向东移动，3：向西移动，4：乘客上车，5：乘客下车
State：500维，（出租车行、出租车列、乘客位置、目的地）
'''
import random
import gym
from time import sleep

env = gym.make('Taxi-v3') #创建出租车游戏环境
env.render() # 用于渲染出当前的智能体以及环境的状态

# 将Q表初始化为一个字典，它存储指定在状态s中执行动作a的值的状态-动作对。
Q = {}
for s in range(env.observation_space.n):
    for a in range(env.action_space.n):
        Q[(s,a)] = 0.0

# epsilon贪心策略函数
def epsilon_greedy(state, epsilon):
    if random.uniform(0,1) < epsilon:
        return env.action_space.sample() # 随机，用epsilon概率探索新动作
    else:
        return max(list(range(env.action_space.n)), key = lambda x: Q[(state,x)]) # 用1-epsilon的概率选择Q表最佳动作
```

```
# 初始化变量
alpha = 0.4 # TD学习率
gamma = 0.999 # 折扣率
epsilon = 0.017 # 贪心策略中epsilon的值
num_episodes = 1000 # 玩几局游戏

# 执行SARSA
for episode in range(num_episodes): # 玩几局游戏
    steps, r = 0, 0  # 每局走多少步，总体奖励
    state = env.reset() # 用于重置环境
    # select the action using epsilon-greedy policy
    action = epsilon_greedy(state, epsilon)
    while True:
        steps += 1 # 每局走多少步
        env.render() # 用于渲染出当前的智能体以及环境的状态
        # then we perform the action and move to the next state, and receive the reward
        nextstate, reward, done, _ = env.step(action)
        # again, we select the next action using epsilon greedy policy
        nextaction = epsilon_greedy(nextstate, epsilon)
        # we calculate the Q value of previous state using our update rule
        Q[(state, action)] += alpha * (reward + gamma * Q[(nextstate, nextaction)] - Q[(state, action)])
        # Q(s, a) <- Q(s, a) + alpha (r + gamma Q(s' , a') - Q(s, a))
        # finally we update our state and action with next action and next state
        action = nextaction # a <- a'
        state = nextstate # s <- s'
        # store the rewards
        r += reward # reward: 即时奖励, r: total reward
        # we will break the loop, if we are at the terminal state of the episode
        if done:
            break
    print(f"Episode: {episode + 1}")  # 玩几局游戏
    print(f"Epochs: {steps}")  # 每局走多少步
```

```
    print(f"State: {state}")
    print(f"Action: {action}")
    print(f"Reward: {reward}")
    print("Total Reward: ", r)
    # sleep(0.01) # 为了让显示变慢，否则画面会非常快
env.close()
```

1.3 结果

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
Episode: 1000
Epochs: 10
State: 0
Action: 0
Reward: 20
Total Reward:  11
```

2. Q-Learning

2.1 算法流程

算法 14.4　Q 学习：一种异策略的时序差分学习算法

输入: 状态空间 $\mathcal{S}$, 动作空间 $\mathcal{A}$, 折扣率 γ, 学习率 α

$\forall s, \forall a$, 随机初始化 $Q(s,a)$; 根据 Q 函数构建策略 π;
repeat
　　初始化起始状态 s;
　　repeat
　　　　在状态 s, 选择动作 $a = \pi^{\epsilon}(s)$;
　　　　执行动作 a, 得到即时奖励 r 和新状态 s';
　　　　$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$; // 更新 Q 函数
　　　　$s \leftarrow s'$;
　　until s 为终止状态;
until $\forall s, a, Q(s,a)$ 收敛;

输出: 策略 $\pi(s) = \arg\max_{a \in |\mathcal{A}|} Q(s,a)$

2.2 Python程序

```
# -*- coding: UTF-8 -*-
# Solving the Taxi Problem using Q Learning
# From: https://github.com/AndyYue1893/Hands-On-Reinforcement-Learning-With-Python
# https://www.cnblogs.com/kailugaji/ - 凯鲁嘎吉 - 博客园
'''
出租车调度
这里有 4 个地点，分别用 4 个字母表示，任务是要从一个地点接上乘客，送到另外 3 个中的一个放下乘客，越快越好。
颜色：蓝色：乘客，红色：乘客的目的地，黄色：空出租车，绿色：出租车满座，其中 “:” 栅栏可以穿越，"|" 栅栏不能穿越
Reward：成功运送一个客人获得 20 分奖励
        每走一步损失 1 分（希望尽快送到目的地）
        没有把客人放到指定的位置，损失 10 分
Action: 0：向南移动，1：向北移动，2：向东移动，3：向西移动，4：乘客上车，5：乘客下车
State：500维，（出租车行、出租车列、乘客位置、目的地）
'''
import random
import gym
from time import sleep

env = gym.make('Taxi-v3') #创建出租车游戏环境
env.render() # 用于渲染出当前的智能体以及环境的状态

# 将Q表初始化为一个字典，它存储指定在状态s中执行动作a的值的状态-动作对。
q = {}
for s in range(env.observation_space.n):
    for a in range(env.action_space.n):
        q[(s,a)] = 0.0

# 定义一个名为update_q_table的函数，根据q学习更新规则更新q值
def update_q_table(prev_state, action, reward, nextstate, alpha, gamma):
    qa = max([q[(nextstate, a)] for a in range(env.action_space.n)]) # 取一个状态-动作对的最大值，并将其存储在一个名为qa的变量中
    # max Q(s', a')
    q[(prev_state, action)] += alpha * (reward + gamma * qa - q[(prev_state, action)]) # 用更新规则更新前一个状态的Q值
    # Q(s, a) <- Q(s, a) + alpha (r + gamma max Q(s', a') - Q(s, a))
```

```
# epsilon贪心策略函数
def epsilon_greedy_policy(state, epsilon):
    if random.uniform(0,1) < epsilon:
        return env.action_space.sample() # 随机，用epsilon概率探索新动作
    else:
        return max(list(range(env.action_space.n)), key = lambda x: q[(state,x)]) # 用1-epsilon的概率选择Q表最佳动作

# 初始化变量
alpha = 0.4  # TD学习率
gamma = 0.999 # 折扣率
epsilon = 0.017 # 贪心策略中epsilon的值
num_episodes = 1000 # 玩几局游戏

# 执行Q-Learning
for episode in range(num_episodes): # 玩几局游戏
    steps, r = 0, 0 # 每局走多少步，总体奖励
    prev_state = env.reset() # 用于重置环境
    while True:
        steps += 1 # 每局走多少步
        env.render() # 用于渲染出当前的智能体以及环境的状态
        # In each state, we select the action by epsilon-greedy policy
        action = epsilon_greedy_policy(prev_state, epsilon)
        # then we perform the action and move to the next state, and receive the reward
        nextstate, reward, done, _ = env.step(action)
        # Next we update the Q value using our update_q_table function
        # which updates the Q value by Q learning update rule
        update_q_table(prev_state, action, reward, nextstate, alpha, gamma)
        # Finally we update the previous state as next state
        prev_state = nextstate # s <- s'
        # Store all the rewards obtained
        r += reward # reward: 即时奖励, r: total reward
        # we will break the loop, if we are at the terminal state of the episode
```

```
        if done:
            break
    print(f"Episode: {episode + 1}") # 玩几局游戏
    print(f"Epochs: {steps}") # 每局走多少步
    print(f"State: {prev_state}")
    print(f"Action: {action}")
    print(f"Reward: {reward}")
    print("Total Reward: ", r)
    # sleep(0.01) # 为了让显示变慢，否则画面会非常快
env.close()
```

2.3 结果

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
Episode: 1000
Epochs: 10
State: 0
Action: 5
Reward: 20
Total Reward:  11
```

3．参考文献

[1] https://github.com/sudharsan13296/Hands-On-Reinforcement-Learning-With-Python

[2] 邱锡鹏，神经网络与深度学习，机械工业出版社，https://nndl.github.io/, 2020.