**强化学习理论学习与代码实现**

# 基于时序差分和Q学习的无模型预测与控制

在强化学习所有的思想中，时序差分（TD）无疑是最核心、最新颖的思想，时序差分结合了蒙特卡洛方法和动态规划法的思想，时序差分和蒙特卡洛方法一样，都是直接从与环境交互得到的经验中进行学习策略，而不需要构建环境动态性的完整模型，同时时序差分又和动态规划方法一样，不需要等到整个片段结束后再进行学习，而是通过自举法，基于已得到的其他状态的估计值来更新当前状态的值函数。

首先关注预测问题，即给定策略情况下，估计其价值函数，对于控制问题，DP、TD和蒙特卡洛法都使用了广义策略迭代的思想。

## 学习目标

* 理解用于预测的TD（0）法；
* 理解在策略控制的SARSA算法；
* 理解离策略控制的Q学习；
* 理解TD法相比较于MC和DP的优点；
* 理解n步TD法是如何将MC和TD法统一起来的；
* 理解TD（λ）前向和后向视角。

## TD预测

TD和MC都是利用经验来求解预测问题。给定策略π的一些经验，两种方法都会更新这些经验中的非终止状态St对于v*π*的估计V。大致来说，蒙特卡洛法需要等到一次访问的回报知道后，再使用该回报作为V(St)的目标。一个适应于非平稳环境的简单每次访问型蒙特卡洛方法可以表示为：

 （1）

而TD方法只需要等待至下一时间步，在t+1时刻，TD立刻就能构造出目标，并使用观察到的奖励Rt+1和估计值V(St+1)进行一次有效的更新。最简单的TD方法在状态转移到St+1并收到Rt+1的奖励时立即做出如下更新：

 （2）

实际上，MC更新的目标是Gt，而TD更新的目标是Rt+1+γV(St+1)。这种TD方法称为TD(0)或单步TD。为TD(0)的完整算法。

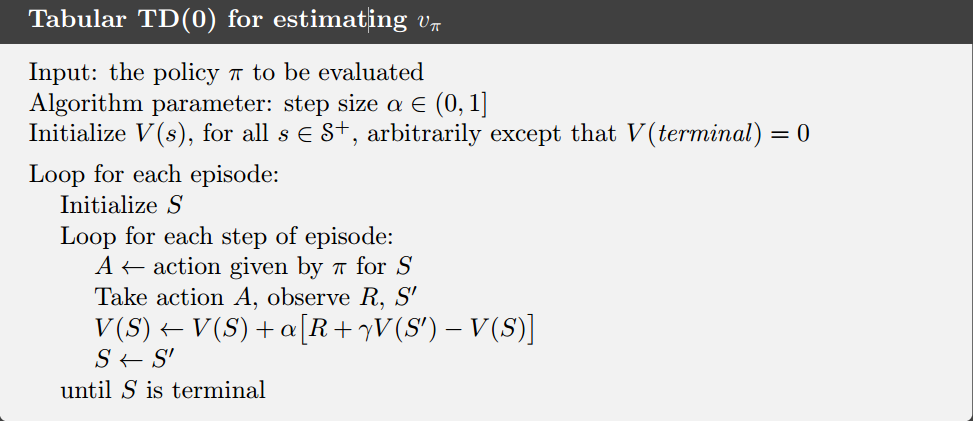


图1 TD（0）

由于TD（0）的更新在某种程度上基于已有的更新，类似于DP，因此也可以将其称为一种自举法。从；；可知

 （3）

大致地讲，MC使用式（3）中的第一个等式作为目标，而DP使用式（3）中的第三个等式作为目标。MC的目标之所以是一个估计值，是因为第一个等式中的期望是未知的，用样本回报替代真实期望值。DP的目标也是一个估计值，不是因为期望值，其会假设由环境模型完整给出，真正的原因是vπ(St+1)是未知的，因而用当前的估计值V(St+1)来替代。TD的同样是一个估计值，原因有两个：它采样得到对第三个等式的期望值，并使用当前估计值V代替真实值vπ，因此TD结合了MC和DP。

## TD预测的优势

相比较于DP，TD一个显而易见的优势在于其不需要一个环境模型，即描述奖励和下一状态概率分布的模型。

另一个显著的优势是，相比于MC，它很自然地运用了一种在线的、完全自增的方式来实现。MC必须等到判断结束，因为只有那时才会知道确切的回报，而TD只需要等到下一时间步即可。

## SARSA：在策略TD控制

现在将TD预测用于控制问题，按照惯例，仍然遵循广义迭代策略（GPI）的模式，只不过这次在评估或预测部分使用TD法。同MC一样，也会面临探索和利用之间的权衡问题，方法同样也划分为两类：在策略和离策略。这部分内容将先展示一下在策略的TD控制。

第一步要学习的是动作值函数而不是状态值函数，特别地，对于在策略法，必须估计出所有状态s以及动作a，在当前行为策略π下的qπ(s,a)，这个估计可以使用于之前学习vπ时完全相同的方法。回想一下，一个片段是一个状态和动作交替出现的序列：



确保桩体值在TD(0)下收敛的定理同样适用于关于动作值的算法上：

 （4）

每当从非终止状态St完成一次转移后就进行一次上面的更新。如果St+1是终止状态，那么Q(St+1,At+1)定义为0。这个更新规则用到了描述这个事件的五元组(St,At,Rt+1,St+1,At+1)中的所有元素，根据这个五元组，将该算法称为Sarsa，其算法流程如图2所示。

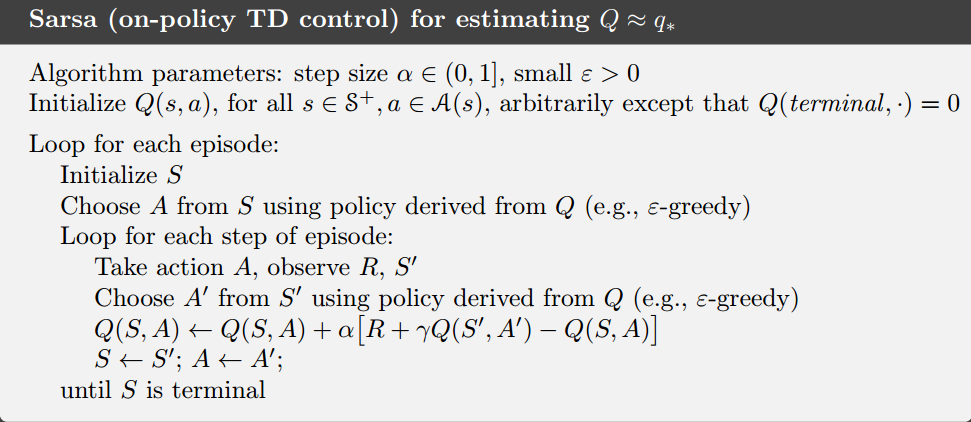


图2 Sarsa算法

## Q学习：离策略TD控制

离策略下的TD控制算法的提出是强化学习早期的一个重要突破。这一算法被称为Q学习（Watkins，1989），其定义为：

 （5）

在这里，待学习的动作值函数Q直接对最优动作值函数q\*进行近似，而与生成决策序列轨迹的行为策略无关，这大大简化了算法的分析，也很早就给出了收敛性证明。正在遵循的行为策略仍会产生影响，它可以决定哪些状态-动作对会被访问和更新。然而，只需要所有的状态-动作对可以持续更新，整个学习过程就能正确收敛。Q学习算法的流程如图3所示。

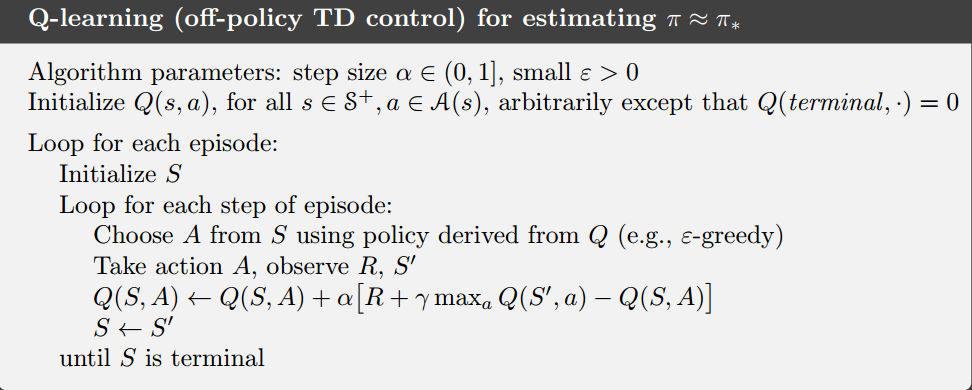


图3 Q学习

## 期望Sarsa

考虑一种与Q学习类似但把对于下一个状态-动作对取最大转换为取期望的学习算法，该算法考虑了当前策略下每个动作的可能性，其更新规则如下：

 （6）

但它又遵循Q学习的模式，给定下一个状态St+1，这个算法确定地向Sarsa的期望方向移动，因此这个算法被称为期望Sarsa。

## n步自举法

n步自举法统一了蒙特卡洛法和时序差分法，是这两种方法更一般的推广，在这个框架下可以更加平滑地切换这两种方法。MC和TD是这个框架中的两种极端的特例，中间方法的性能一般要比这两种极端方法好。

### n步TD预测

MC方法根据从某个状态开始到片段结束所观察到的奖励整个序列，对该状态进行值估计。而单步TD方法则是只根据下一即时报酬，在下一个状态的价值估计值的基础上进行自举更新，这里的价值估计值代理了后面所有剩余时刻的累积奖励。因此可以很自然地想到，如果执行两步、三步等既不是单步又没有到终止状态的时间步？如图4所示。

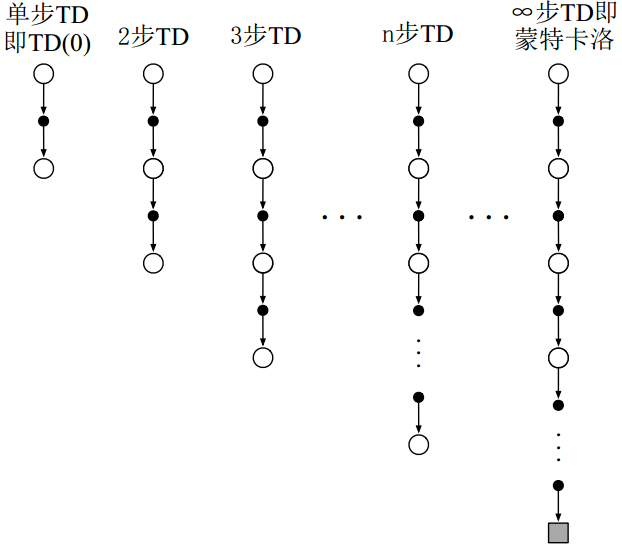


图4 n步法的回溯图

n步更新的方法仍然属于TD方法，因为在这些方法中，前面状态估计值会根据它与后继状态的估计值的差异进行更新，不同的是这里的后继状态是n步后状态，而不是后一步后的状态。将时序差分扩展到n步的方法称为n步TD。

根据前面的知识，在MC的更新中，vπ(St)的估计值会沿着完整回报的方向进行更新，即

 （7）

但在单步TD中，其更新的目标是即时奖励加上后一状态的折扣估计值，称为单步回报：

 （8）

类似地，可以扩展到两步TD，其更新的目标是两步回报：

 （9）

类似地，任意n步更新的目标是n步回报：

 （10）

其中，*n*≥1，0≤t≤*T*-*n*。

***注意，计算n步回报（n>1）时会涉及诸多未来奖励和状态，而当从时刻t转移到t+1时却无法获取这些奖励和状态。n步回报只有在得知Rt+n和Vt+n-1后才能计算，而这些值都只能在t+n时刻才能得到。一个比较自然的基于n步回报的状态值函数学习算法如下：***

 （11）

而对于其他状态（s≠St）的价值估计值保持不变：Vt+n(S)=Vt+n-1(S)。这个算法被称为n步TD。注意，在最开始的n-1个时刻，不会更新价值函数，为了弥补这个缺失，在片段结束后还需执行对应次数的更新。完整的伪代码如下：

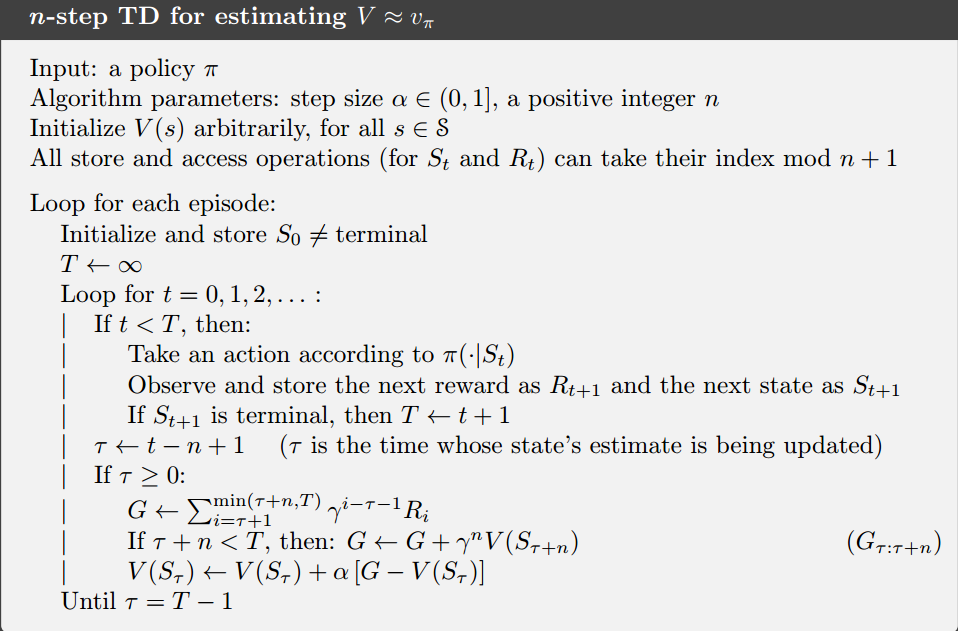


图5 n步TD

### n步Sarsa

这部分内容将展示n步方法是如何和Sarsa直接结合来产生在策略TD控制方法的，这种n步版本的Sarsa称为n步Sarsa。

该方法的核心思想是将状态替换为状态-动作对，然后使用ε贪婪策略，n步Sarsa的回溯图如图6所示。利用类似的方式，可以得到根据估计动作值定义的n步方法的回报：

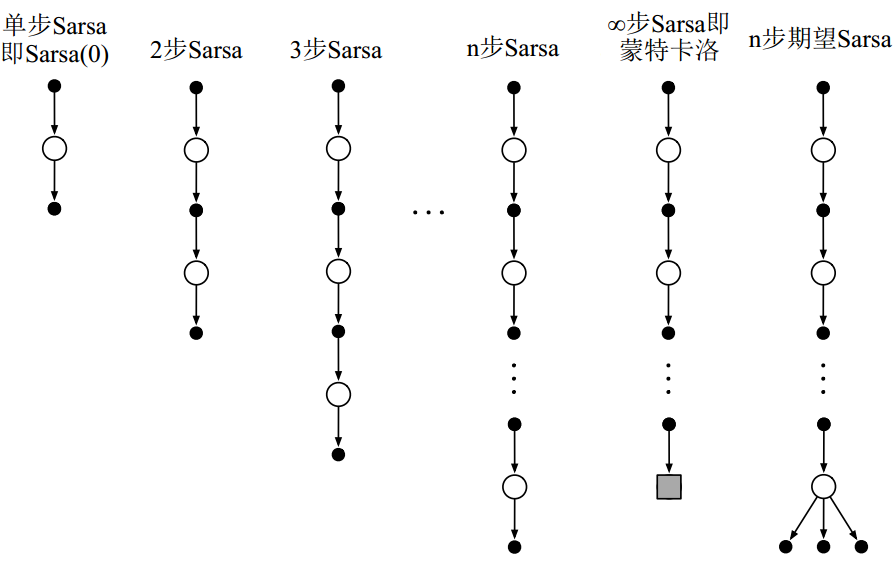


图6 状态-动作值函数的n步回溯图

 （12）

状态-动作值的更新如下：

 （13）

除了上面更新的状态-动嘴对以外，所有其他状态-动作对值都保持不变，即对于所有满足s≠St或a≠at的s，a来说，有Qt+n(s,a)=Qt+n-1(s,a)，这就是n步Sarsa算法，伪代码如图7所示。

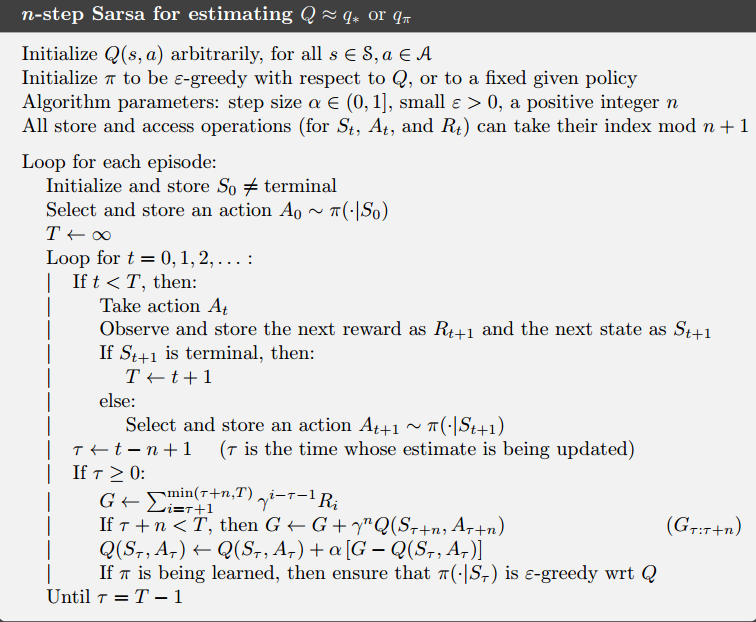


图7 n步Sarsa

### n步离策略学习

离策略学习是在学习策略π时，智能体遵循的却是另一个策略b的学习方法。通常π是针对当前动作值函数估计值的贪婪策略，而b是一个更具探索性的策略，例如ε贪婪策略。为了能够使用从策略b中得到的数据，必须考虑两种策略之间的不同，用它们对应动作的相对概率。在n步方法中，回报根据n步建立，因此感兴趣的也是这n步相对概率，可以简单地用ρt:t+n-1来加权：

 （14）

其中ρt:t+n-1为重要度采样，

 （15）

同样，n步Sarsa的离策略版本如下：

 （16）

注意这里的重要度采样率，其起点和终点比式（14）中的都要晚一步，这是因为在这里我们更新的是状态-动作对，这时并不关心这些动作被选择的概率有多大，既然已经确定了这个动作，那么想要的是充分地学习发生的事情，这个学习过程会使用基于后继动作计算出的重要度采样加权，下面为其伪代码：

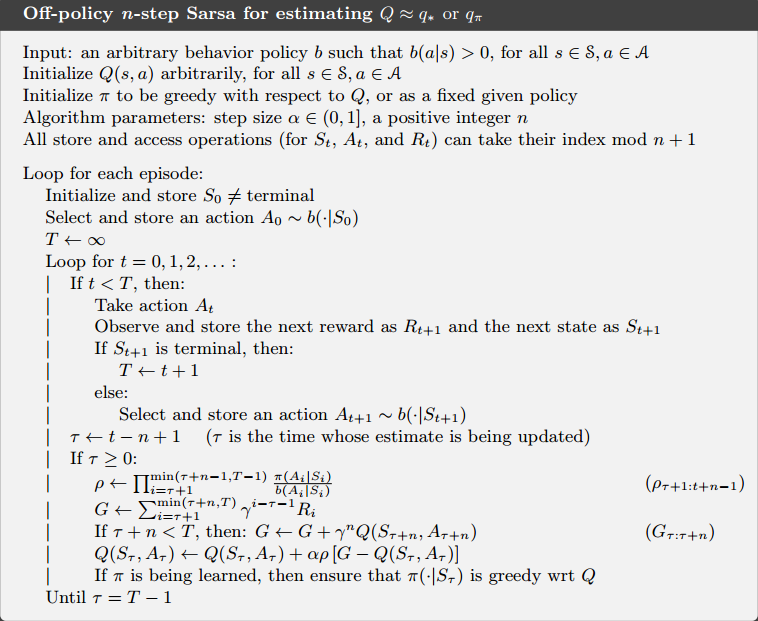


图8 离策略下的n步Sarsa

## 代码练习

### SARSA

# -\*- coding: utf-8 -\*-

"""

Created on Tue Nov 5 17:03:40 2019

@author: hba

"""

import gym

import itertools

import matplotlib

import numpy as np

import pandas as pd

import sys

if "../" not in sys.path:

sys.path.append("../")

from collections import defaultdict

from Lib.envs.windy\_gridworld import WindyGridworldEnv

from Lib import plotting

matplotlib.style.use('ggplot')

env = WindyGridworldEnv()

def make\_epsilon\_greedy\_policy(Q,epsilon,nA):

"""

基于给定的Q值函数和epsilon创建epsilon贪婪策略

参数：

Q:将状态映射成动作值函数的字典。每个值都是一个长度为nA的numpy数组。

epsilon：选择随机动作的概率，为0与1之间的浮点数。

nA：环境中的动作数。

返回值：

返回一个函数，该函数的输入为观察即状态，并以numpy数组（长度为nA）的形式返回每个动作的概率。

"""

def policy\_fn(obseration):

A=np.ones(nA,dtype=float)\*epsilon/nA

best\_action=np.argmax(Q[obseration])

A[best\_action]+=(1.0-epsilon)

return A

return policy\_fn

def sarsa(env, num\_episodes, discount\_factor=1.0, alpha=0.5, epsilon=0.1):

"""

sarsa算法：在策略TD控制—寻找最优epsilon贪婪策略。

参数:

env: OpenAI环境.

num\_episodes: 迭代次数.

discount\_factor: Gamma折扣因子.

alpha: TD学习率.

epsilon: 采样随机动作的概率，为0~1的浮点数.

返回值:

一个元组(Q, episode\_lengths)吗，其中Q是最优动作值函数，将状态映射为动作值.

"""

# 最终的动作值函数.

Q = defaultdict(lambda: np.zeros(env.action\_space.n))

# 跟踪有效统计

stats = plotting.EpisodeStats(

episode\_lengths=np.zeros(num\_episodes),

episode\_rewards=np.zeros(num\_episodes))

# 遵循的策略

policy = make\_epsilon\_greedy\_policy(Q, epsilon, env.action\_space.n)

for i\_episode in range(num\_episodes):

#打印当前代，用于调试

if (i\_episode + 1) % 100 == 0:

print("\rEpisode {}/{}.".format(i\_episode + 1, num\_episodes), end="")

sys.stdout.flush()

# Reset the environment and pick the first action

state = env.reset()

action\_probs = policy(state)

action = np.random.choice(np.arange(len(action\_probs)), p=action\_probs)

# One step in the environment

for t in itertools.count():

#执行一步

next\_state, reward, done, \_ = env.step(action)

# 选取下一动作

next\_action\_probs = policy(next\_state)

next\_action = np.random.choice(np.arange(len(next\_action\_probs)), p=next\_action\_probs)

stats.episode\_rewards[i\_episode] += reward

stats.episode\_lengths[i\_episode] = t

# TD更新

td\_target = reward + discount\_factor \* Q[next\_state][next\_action]

td\_delta = td\_target - Q[state][action]

Q[state][action] += alpha \* td\_delta

if done:

break

action = next\_action

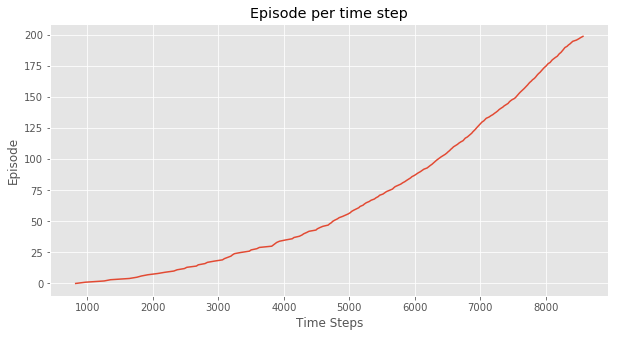
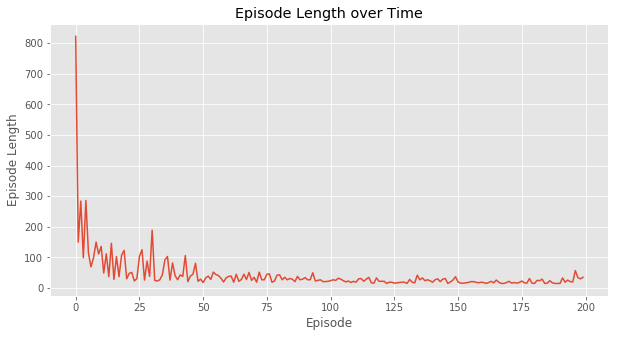
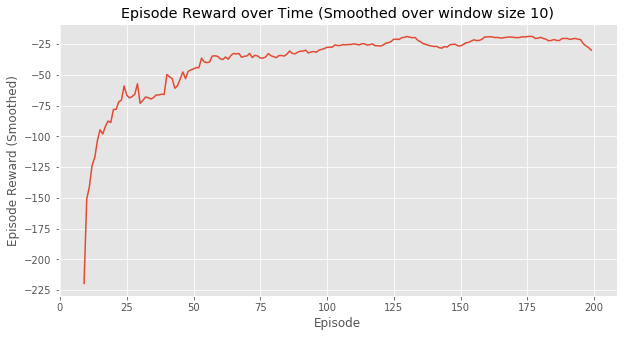
state = next\_state

return Q, stats

Q, stats = sarsa(env, 200)

plotting.plot\_episode\_stats(stats)

结果如下：



### Q学习

# -\*- coding: utf-8 -\*-

"""

Created on Tue Nov 5 17:03:40 2019

@author: hba

"""

import gym

import itertools

import matplotlib

import numpy as np

import pandas as pd

import sys

if "../" not in sys.path:

sys.path.append("../")

from collections import defaultdict

from Lib.envs.cliff\_walking import CliffWalkingEnv

from Lib import plotting

matplotlib.style.use('ggplot')

env = CliffWalkingEnv()

def make\_epsilon\_greedy\_policy(Q,epsilon,nA):

"""

基于给定的Q值函数和epsilon创建epsilon贪婪策略

参数：

Q:将状态映射成动作值函数的字典。每个值都是一个长度为nA的numpy数组。

epsilon：选择随机动作的概率，为0与1之间的浮点数。

nA：环境中的动作数。

返回值：

返回一个函数，该函数的输入为观察即状态，并以numpy数组（长度为nA）的形式返回每个动作的概率。

"""

def policy\_fn(obseration):

A=np.ones(nA,dtype=float)\*epsilon/nA

best\_action=np.argmax(Q[obseration])

A[best\_action]+=(1.0-epsilon)

return A

return policy\_fn

def q\_learning(env, num\_episodes, discount\_factor=1.0, alpha=0.5, epsilon=0.1):

"""

Q-Learning算法：离策略TD控制—遵循epsilon贪婪策略的同时学习最优贪婪策略。

参数:

env: OpenAI环境.

num\_episodes: 迭代次数.

discount\_factor: Gamma折扣因子.

alpha: TD学习率.

epsilon: 采样随机动作的概率，为0~1的浮点数.

返回值:

一个元组(Q, episode\_lengths)吗，其中Q是最优动作值函数，将状态映射为动作值.

"""

# 最终的动作值函数.

Q = defaultdict(lambda: np.zeros(env.action\_space.n))

# 跟踪有效统计

stats = plotting.EpisodeStats(

episode\_lengths=np.zeros(num\_episodes),

episode\_rewards=np.zeros(num\_episodes))

# 遵循的策略

policy = make\_epsilon\_greedy\_policy(Q, epsilon, env.action\_space.n)

for i\_episode in range(num\_episodes):

#打印当前代，用于调试

if (i\_episode + 1) % 100 == 0:

print("\rEpisode {}/{}.".format(i\_episode + 1, num\_episodes), end="")

sys.stdout.flush()

# Implement this!

state=env.reset()

for t in itertools.count():

action\_probs =policy(state)

action = np.random.choice(np.arange(len(action\_probs)), p=action\_probs)

next\_state,reward,done,\_=env.step(action)

stats.episode\_rewards[i\_episode] += reward

stats.episode\_lengths[i\_episode] = t

next\_best\_action=np.argmax(Q[next\_state])

td\_target=reward+discount\_factor\*Q[next\_state][next\_best\_action]

td\_error=td\_target-Q[state][action]

Q[state][action]+=alpha\*td\_error

if done:

break

state=next\_state

return Q, stats

Q, stats = q\_learning(env, 500)

plotting.plot\_episode\_stats(stats)

结果为：

