《博弈论与强化学习》课程设计报告

课程设计 1：SARSA 和 Q-learning 方法实践

姓名	李昀哲		学号	20123101
学院	计算机工程与科学学院		专业	智能科学与技术
得分	代码得分(30)		总分	
	汇报得分(30)			
	报告得分(40)			

指导教师评语：

课程设计目的及内容：

在实际问题中，马尔可夫决策过程 MDP 模型是未知的，或是因为太大或太复杂而无法使用，智能体只能通过和环境交互获得 trajectories，不能直接获得转移矩阵和奖励函数。因此，我们需要通过不学习 MDP，从经验中直接学习价值函数和策略，也就是模型无关的强化学习（Model-free RL）。和之前类似的是，需要解决的问题同样是估计值函数的预测问题 Prediction，和优化策略的控制问题 Control。时序差分学习 Temporal-Difference Learning 是一种解决方案，从经验序列中直接学习，不需要知道 MDP 的转移矩阵和奖励，同时 TD Learning 通过 bootstrapping 从不完整的 episode 中学习。它是动态规划和蒙特卡洛法的折中。TD Learning 通过策略π下得到的经验在线学习v_π，更新当前预测值使之接近估计累计奖励。

由 TD Learning 引出两种 model-free 的强化学习，分别是在线策略学习 On-policy 的 Sarsa：利用策略π的经验采样不断学习改进策略π；和离线策略学习 Off-policy 的 Q Learning：利用另一个策略μ的经验采样不断学习改进策略π。

Sarsa 算法，即 State-Action-Reward-State-Action 的元组，**根据策略π收集经验，再根据经验去学习策略π**，使用 $Q(S_{t+1}, A_{t+1})$来近似状态的期望值 $V(S_{t+1})$。采取并不是最优的动作，目的是为了探索所有的动作，然后减少探索。在更新 Q-table 前先将下一个要执行动作用当前策略选出来。策略改进使用ε-greedy 策略改进。

Q learning 使用两种不同的策略，一个策略用于学习并成为最佳策略，另一个策略用于探索并产生 trajectories，**根据另一个策略μ收集经验，再根据经验学习策略π**，在更新 Q-table 前选出的动作并不真正执行。

本次课程设计旨在比较 Saras 和 Q-learning 两种算法的学习效果，进而对比 on-policy 和 off-policy 两种强化学习类别的学习效果，进一步理解两种方法的异同和适用场景。

课程设计过程中遇到的问题以及解决方法？

问题 1：因为 on-policy 和 off-policy 两种类别的学习都基于 TD Learning，故重新对 TD Learning 进行理解，尝试区分 TD Learning 和 MC 相比的异同。

他们的目标是相同的，都是从策略π下的经验片段学习V^{π}；MC 的实现方法是增量地进行每次蒙特卡洛过程，更新值函数 $V(S_t)$使之接近准确累积奖励；而 TD 则是更新 $V(S_t)$使之接近估计累积奖励。同时，在学习的阶段、学习的内容、适用场景、方差、初值的选择上均有差异，如下表所示。

表 1 TD Learning 和 MC 对比

	TD	**MC**
学习条件	执行完每一步后	执行完一个完整的 episode 后
序列需求	可以不完整	必须完整
适用场景	连续、无终止条件的场景，在，马尔科夫环境中更高效	只能适用有终止条件，在非马尔科夫环境更高效
方差	低	高
偏差	有	无
对初始值敏感度	相对更高	不敏感
其他特点	最终收敛到$V^{\pi}(S_t)$	易于理解和适用

问题 2：两种类型的学习效果数据并不容易看出差异，且数据震荡较大，难以趋于平稳。

调整记录数据的间隔，从每 1000 记录一次变为每 100 记录一次；同时调整学习率和每一个 step 减少的学习速率；调整最大迭代次数，尽可能使结果趋于收敛。

问题 3：为什么 SARSA 算法学习效果的数据即使在迭代较多次数后震荡幅度仍很大？

这个问题仍要回归到 SARSA 的策略提升方式，不同于 Q Learning，在线策略时序差分控制 Sarsa 的策略改进仅仅使用ε-greedy 的方式，每一步都会选择当前最优的方向，因此在正常情况下都会出现较大幅度的震荡。

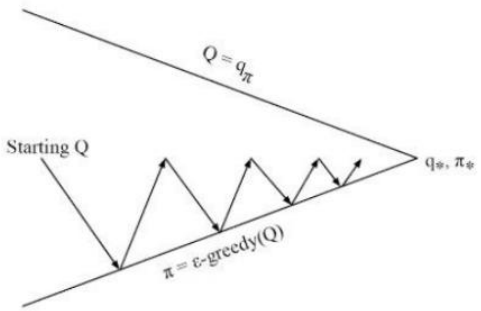


图 1 SARSA 算法迭代过程

本次课程设计的体会

本次课程设计，通过编码，实现了在 cliff walking task 下对 Q Learning 和 SARSA 两种算法进行了实现和测试，加深两种算法理论理解的同时，实践其在实际问题中的学习效果，并改变相关参数如：最大迭代次数、学习减少速率、学习率等。通过实验，得到 Q Learning 在大部分情况下可以获得比 SARSA 更好的得分，如表 2 所示，证明了 Off-policy 使用的两种策略更具优势。

表 2 不同迭代次数下两种算法得分

Iter_max	5000	20000
Sarsa	-185.15	-155.28
Q Learning	**-180.92**	**-141.52**

在对比两种算法迭代过程中的数据时发现，SARSA 的震荡幅度明显大于 Q Learning，印证了其仅仅使用贪婪策略进行改进产生的弊端，不过，即使得分不是最优的，但至少是一条可行的路径，是不使用 max 操作的结果；而 Q Learning 使用 max 操作能收敛到最优路径。

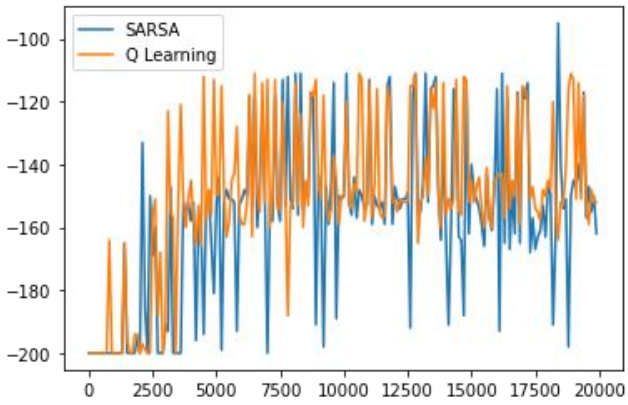


图 2 最大迭代次数为 20000 时两种算法学习效果对比

思考题：

思考题 1：在线策略(On-policy)与离线策略(Off-policy)学习效果比较

以 Sarsa 算法为代表的 on-policy 往往收敛到距离悬崖边比较远的路径，称之为安全路径，虽然不是最佳的，但是是可行的；而 off-policy 的 Q Learning 算法往往收敛到距离悬崖边较近的贪婪路径，也是最优的路径。在迭代过程中，Sarsa 相对保守，没有使用 max 操作的过程；而 Q Learning 则使用贪婪的策略进行改进，会使用 max 操作，能更好地平衡 exploration 和 exploitation，遵循探索策略时，学习最佳策略。从上述课程设计结果表 2 来看，平均的得分也是 Q Learning 更高，符合理论。

```
------ using SARSA Learning ---
Iteration #1 -- Total reward = -200.
Iteration #201 -- Total reward = -200.
Iteration #401 -- Total reward = -200.
Iteration #601 -- Total reward = -200.
Iteration #801 -- Total reward = -167.
Iteration #1001 -- Total reward = -200.
Iteration #1201 -- Total reward = -200.
Iteration #1401 -- Total reward = -156.
Iteration #1601 -- Total reward = -200.
Iteration #1801 -- Total reward = -194.
Iteration #2001 -- Total reward = -200.
Iteration #2201 -- Total reward = -197.
Iteration #2401 -- Total reward = -200.
Iteration #2601 -- Total reward = -154.
Iteration #2801 -- Total reward = -151.
Iteration #3001 -- Total reward = -195.
Iteration #3201 -- Total reward = -165.
Iteration #3401 -- Total reward = -159.
Iteration #3601 -- Total reward = -191.
Iteration #3801 -- Total reward = -159.
Iteration #4001 -- Total reward = -160.
Iteration #4201 -- Total reward = -155.
Iteration #4401 -- Total reward = -156.
Iteration #4601 -- Total reward = -157.
Iteration #4801 -- Total reward = -156.
Average score of solution =  -185.15
```

```
----- using Q Learning -----
Iteration #1 -- Total reward = -200.
Iteration #201 -- Total reward = -200.
Iteration #401 -- Total reward = -200.
Iteration #601 -- Total reward = -200.
Iteration #801 -- Total reward = -200.
Iteration #1001 -- Total reward = -200.
Iteration #1201 -- Total reward = -200.
Iteration #1401 -- Total reward = -165.
Iteration #1601 -- Total reward = -166.
Iteration #1801 -- Total reward = -200.
Iteration #2001 -- Total reward = -200.
Iteration #2201 -- Total reward = -200.
Iteration #2401 -- Total reward = -200.
Iteration #2601 -- Total reward = -159.
Iteration #2801 -- Total reward = -200.
Iteration #3001 -- Total reward = -199.
Iteration #3201 -- Total reward = -162.
Iteration #3401 -- Total reward = -200.
Iteration #3601 -- Total reward = -192.
Iteration #3801 -- Total reward = -200.
Iteration #4001 -- Total reward = -200.
Iteration #4201 -- Total reward = -200.
Iteration #4401 -- Total reward = -200.
Iteration #4601 -- Total reward = -164.
Iteration #4801 -- Total reward = -173.
Average score of solution =  -180.92
```

图 3 最大迭代次数为 5000 时实验结果

代码附录

环境类

```
import time
import random
```

```
class Env():
    def __init__(self, length, height):
        # define the height and length of the map
        self.length = length
        self.height = height
        # define the agent's start position
        self.x = 0
        self.y = 0

    def render(self, frames=50):
        for i in range(self.height):
            if i == 0: # cliff is in the line 0
                line = ['S'] + ['x']*(self.length - 2) + ['T'] # 'S':start, 'T':terminal, 'x':the cliff
            else:
                line = ['.'] * self.length
            if self.x == i:
                line[self.y] = 'o' # mark the agent's position as 'o'
            print(''.join(line))
        print('\033['+str(self.height+1)+'A')  # printer go back to top-left
        time.sleep(1.0 / frames)

    def step(self, action):
        """4 legal actions, 0:up, 1:down, 2:left, 3:right"""
        change = [[0, 1], [0, -1], [-1, 0], [1, 0]]
        self.x = min(self.height - 1, max(0, self.x + change[action][0]))
        self.y = min(self.length - 1, max(0, self.y + change[action][1]))

        states = [self.x, self.y]
        reward = -1
        terminal = False
        if self.x == 0: # if agent is on the cliff line "SxxxxxT"
            if self.y > 0: # if agent is not on the start position
                terminal = True
                if self.y != self.length - 1: # if agent falls
                    reward = -100
        return reward, states, terminal

    def reset(self):
        self.x = 0
        self.y = 0
```

定义ε-greedy 的迭代和更新动作

```
class Q_table():
    def __init__(self, length, height, actions=4, alpha=0.1, gamma=0.9):
        self.table = [0] * actions * length * height # initialize all Q(s,a) to zero
        self.actions = actions
        self.length = length
        self.height = height
        self.alpha = alpha
        self.gamma = gamma

    def _index(self, a, x, y):
        """Return the index of Q([x,y], a) in Q_table."""
        return a * self.height * self.length + x * self.length + y

    def _epsilon(self):
        return 0.1

    def take_action(self, x, y, num_episode):
        # epsilon-greedy
        if random.random() < self._epsilon():
            return int(random.random() * 4)
        else:
            actions_value = [self.table[self._index(a, x, y)] for a in range(self.actions)]
            return actions_value.index(max(actions_value))

    def max_q(self, x, y):
        actions_value = [self.table[self._index(a, x, y)] for a in range(self.actions)]
        return max(actions_value)

    def update(self, a, s0, s1, r, is_terminated):
        # both s0, s1 have the form [x,y]
        q_predict = self.table[self._index(a, s0[0], s0[1])]
        if not is_terminated:
            q_target = r + self.gamma * self.max_q(s1[0], s1[1])
        else:
            q_target = r
        self.table[self._index(a, s0[0], s0[1])] += self.alpha * (q_target - q_predict)
```

悬崖行走

```
def cliff_walk():
    env = Env(length=12, height=4)
    table = Q_table(length=12, height=4)
    for num_episode in range(3000):
        # within the whole learning process
        episodic_reward = 0
        is_terminated = False
        s0 = [0, 0]
        while not is_terminated:
            # within one episode
            action = table.take_action(s0[0], s0[1], num_episode)
            r, s1, is_terminated = env.step(action)
            table.update(action, s0, s1, r, is_terminated)
            episodic_reward += r
            # env.render(frames=100)
            s0 = s1
        if num_episode % 20 == 0:
            print("Episode: {}, Score: {}".format(num_episode, episodic_reward))
        env.reset()
```

```
cliff_walk()
```

```
Episode: 0, Score: -100
Episode: 20, Score: -243
Episode: 40, Score: -195
Episode: 60, Score: -100
Episode: 80, Score: -38
Episode: 100, Score: -150
Episode: 120, Score: -56
Episode: 140, Score: -49
Episode: 160, Score: -39
Episode: 180, Score: -109
Episode: 200, Score: -135
Episode: 220, Score: -23
Episode: 240, Score: -25
Episode: 260, Score: -21
Episode: 280, Score: -100
Episode: 300, Score: -28
Episode: 320, Score: -24
Episode: 340, Score: -13
Episode: 360, Score: -15
```

导入强化学习 gym 库，初始化参数

```
import numpy as np

import gym
from gym import wrappers

off_policy = False # if True use off-policy q-learning update, if False, use on-policy SARSA update

n_states = 40
iter_max = 20000

initial_lr = 1 # Learning rate
min_lr = 0.003
gamma = 1.0
t_max = 10000
eps = 0.1

def run_episode(env, policy=None, render=False):
    obs = env.reset()
    total_reward = 0
    step_idx = 0
    for _ in range(t_max):
        if render:
            env.render()
        if policy is None:
            action = env.action_space.sample()
        else:
            a,b = obs_to_state(env, obs)
            action = policy[a][b]
        obs, reward, done, _ = env.step(action)
        total_reward += gamma ** step_idx * reward
        step_idx += 1
        if done:
            break
    return total_reward
```

```python
def obs_to_state(env, obs):
    """ Maps an observation to state """
    # we quantify the continous state space into discrete space
    env_low = env.observation_space.low
    env_high = env.observation_space.high
    env_dx = (env_high - env_low) / n_states
    a = int((obs[0] - env_low[0])/env_dx[0])
    b = int((obs[1] - env_low[1])/env_dx[1])
    a = a if a < n_states else n_states - 1
    b = b if b < n_states else n_states - 1
    return a, b
```

主函数

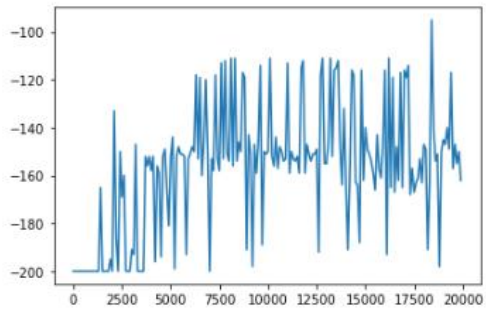
```python
if __name__ == '__main__':
    env_name = 'MountainCar-v0'
    env = gym.make(env_name)
    env.seed(0)
    np.random.seed(0)
    if off_policy == True:
        print ('----- using Q Learning -----')
    else:
        print('------ using SARSA Learning ---')

    q_table = np.zeros((n_states, n_states, 3))
    it_cnt = []
    reward_cnt = []
    for i in range(iter_max):
        obs = env.reset()
        total_reward = 0
        ## eta: learning rate is decreased at each step
        eta = max(min_lr, initial_lr * (0.95 ** (i//100)))
        for j in range(t_max):
            a, b = obs_to_state(env, obs)
            if np.random.uniform(0, 1) < eps:
                action = np.random.choice(env.action_space.n)
            else:
                action = np.argmax(q_table[a][b])
            obs, reward, done, _ = env.step(action)
            total_reward += reward
            # update q table
            a_, b_ = obs_to_state(env, obs)
            if off_policy == True:
                # use q-learning update (off-policy learning)
                q_table[a][b][action] = q_table[a][b][action] + eta * (reward + gamma *  np.max(q_table[a_][b_]) - q_table[a][b]
            else:
                # use SARSA update (on-policy learning)
                # epsilon-greedy policy on Q again
                if np.random.uniform(0,1) < eps:
                    action_ = np.random.choice(env.action_space.n)
                else:
                    action_ = np.argmax(q_table[a_][b_])
                q_table[a][b][action] = q_table[a][b][action] + eta * (reward + gamma *  q_table[a_][b_][action_] - q_table[a][b]
            if done:
                break
        if i % 100 == 0:
#             print('Iteration #%d -- Total reward = %d.' %(i+1, total_reward))
            it_cnt.append(i)
            reward_cnt.append(total_reward)
    solution_policy = np.argmax(q_table, axis=2)
    solution_policy_scores = [run_episode(env, solution_policy, False) for _ in range(100)]
    print("Average score of solution = ", np.mean(solution_policy_scores))
    # Animate it
    for _ in range(2):
        run_episode(env, solution_policy, True)
    env.close()
```

数据分析

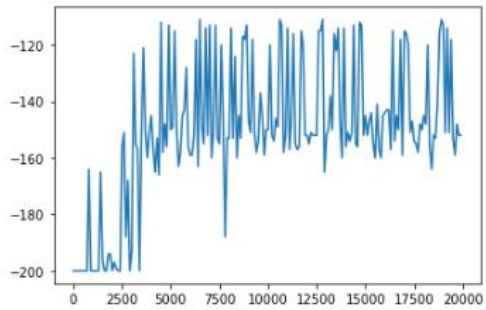
```
plt.plot(it_cnt, reward_cnt)
```

```
[<matplotlib.lines.Line2D at 0x2115cbe0988>]
```



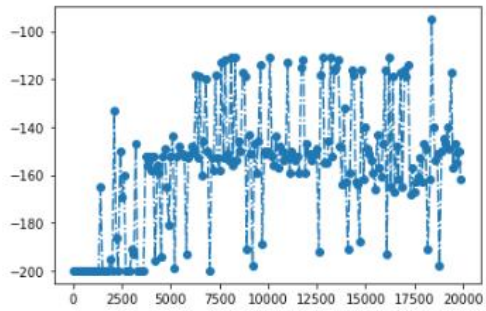

```
plt.plot(Q_it, Q_reward)
```

```
[<matplotlib.lines.Line2D at 0x2115ccc4288>]
```




```
SARSA_it, SARSA_reward = it_cnt, reward_cnt
plt.plot(SARSA_it, SARSA_reward, '-.o')
```

```
[<matplotlib.lines.Line2D at 0x2115cfadfc8>]
```




```
plt.plot(SARSA_it, SARSA_reward, label='SARSA')
plt.plot(Q_it, Q_reward, label='Q Learning')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2115eadadc8>
```