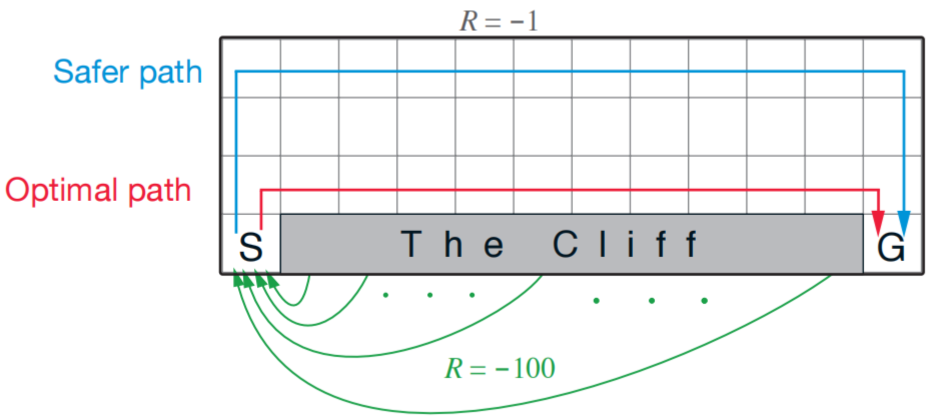
日期	科目	班级	姓名	学号
2023 年 3 月 22 日	强化学习	强基数学 002	吴天阳	2204210460

第六章作业

题目 1. 例 6.6、练习 6.12：在悬崖边上行走

如右图所示是一个 4×12 的网格图，起点在左下角，终点在右下角，最下面一行除去左右端点外均为悬崖. 每一幕开始时，智能体从起点开始移动，每一步有上下左右四个方向可以选择，如果到达悬崖或达到终点，结束当前幕. 当智能体掉下悬崖时收益为 -100，其他每步收益均为 -1，使用 sarsa 和 Q Learning 求解该问题，比较在不同 ε 下，sarsa 和 Q Learning 的策略有何不同.



解答. 设置步长 $\alpha=0.9$，折扣率为 $\gamma=1$，分别测试 $\varepsilon=0.1,0.01,0$ 下 sarsa 和 Q Learning 在每一幕下收益之和的变换情况，其中 $\varepsilon=0$ 即使用贪心方式选择策略.

通过平均 1000 次试验结果，得到以下收益之和变换情况（如图 2 所示），并输出不同 ε 下的最优策略（如图 1 所示），可以发现，在 $\varepsilon=0$ 时，两者使用了相同的最优贪心策略，并且收益之和曲线几乎完全重合，即在悬崖边上行走；但当 ε 增大时，Q Learning 仍采用最优策略，但是 sarsa 逐渐转换为安全策略，靠远离悬崖的路线上行走.

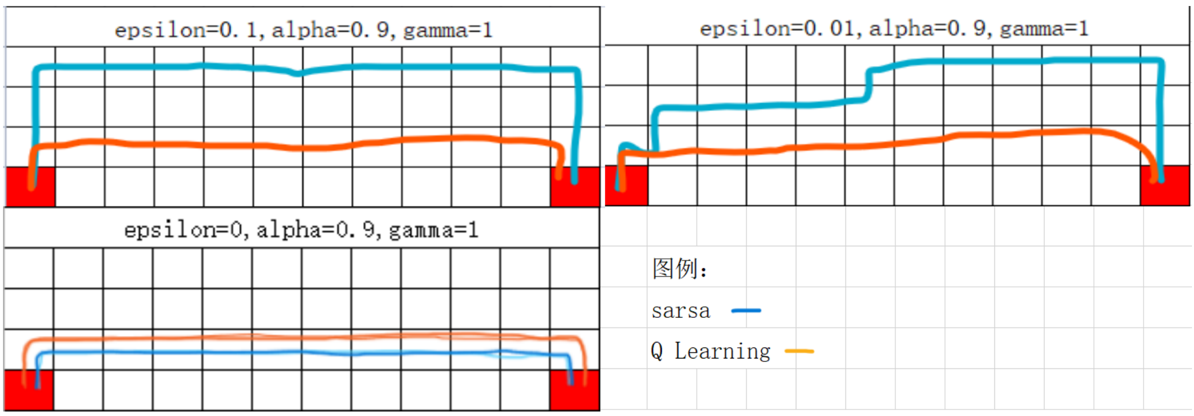


图 1: 不同 ε 下 sarsa 和 Q Learning 策略比较图

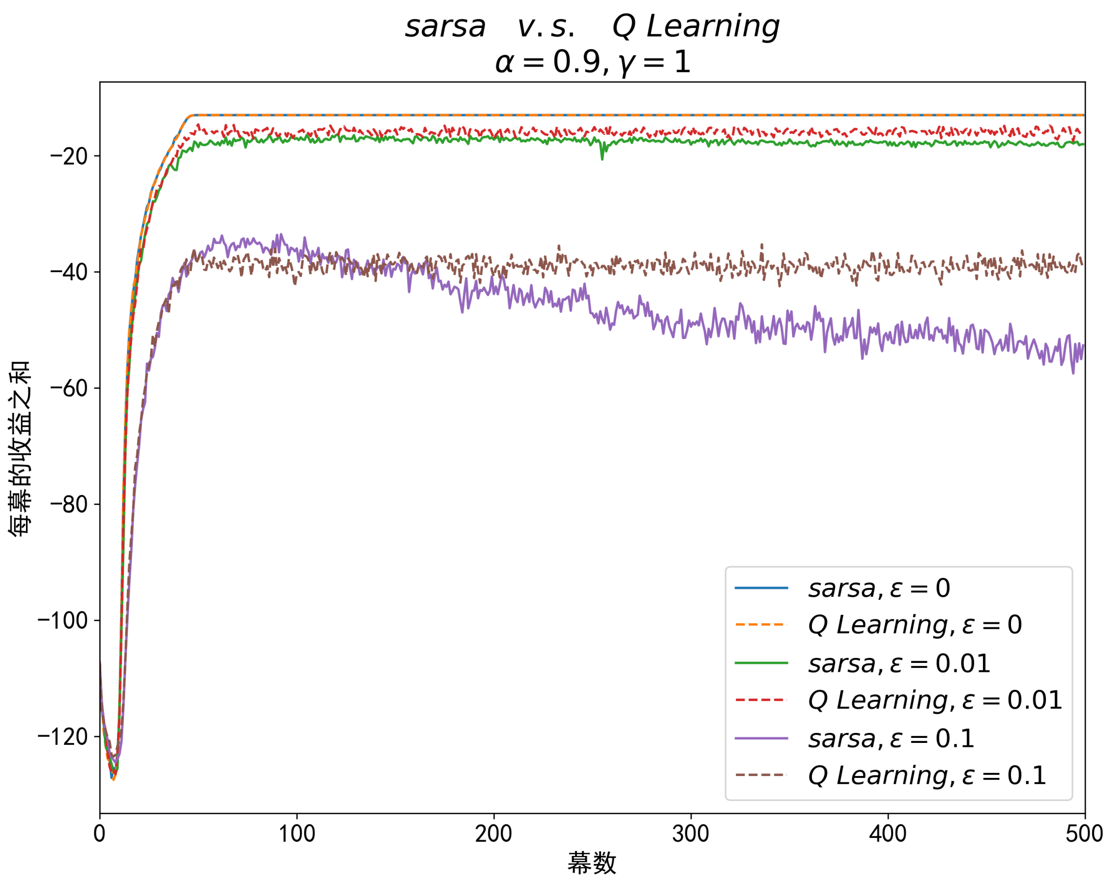


图 2: 不同 ε 下策略收益变换情况

源代码:

```python
# -*- coding: utf-8 -*-
'''
@File    : main.py
@Time    : 2023/03/21 19:43:45
@Author  : wty-yy
@Version : 1.0
@Blog    : https://wty-yy.space/
@Desc    : 《强化学习》中文书第 130 页例 6.6 复现 (epsilon=0.1), 并完成练习 6.12(epsilon=0)
'''

from tqdm import tqdm
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

config = {  # matplotlib 绘图配置
    "figure.figsize": (6, 6),  # 图像大小
    "font.size": 16, # 字号大小
    "font.sans-serif": ['SimHei'],   # 用黑体显示中文
    'axes.unicode_minus': False # 显示负号
}
```

```python
plt.rcParams.update(config)

ACTION_TO_DIRECTION = [np.array(x) for x in ((0, 1), (0, -1), (1, 0), (-1, 0))]

def check_available_position(position):
    return 0 <= position[0] < 4 and 0 <= position[1] < 12

def position_action_iterator():
    for x in range(4):
        for y in range(12):
            position = np.array((x, y))
            for action in range(4):
                yield position, action

def plot_savefig(title: str, fname: str):
    plt.xlabel("幕数")
    plt.ylabel("每幕的收益之和")
    plt.title(title)
    plt.legend()
    plt.xlim(0, 500)
    plt.tight_layout()
    plt.savefig(fname, dpi=300)
    plt.show()

class WalkingEnvironment:
    def __init__(self) -> None:
        self.position = np.array(())

    def step(self, action: int):
        direction = ACTION_TO_DIRECTION[action]
        position_ = self.position + direction
        assert(check_available_position(position_))
        reward = -100 if position_[0] == 3 and 1 <= position_[1] < 11 else -1
        terminal = True if (position_[0] == 3 and position_[1] == 11) or (reward == -100) else False
        # self.position = self.reset() if reward == -100 else position_
        self.position = position_
        return reward, self.position, terminal

    def reset(self):
        self.position = np.array((3, 0))
        return self.position

class Agent:
    alpha = None
    def __init__(self, epsilon=0.1, gamma=1, seed=42, episode=500, average_times=1000, is_plot=True) -> None:
        self.epsilon = epsilon
        self.gamma = gamma
        np.random.seed(seed)
        self.episode = episode
        self.average_times = average_times
        self.is_plot = is_plot
```

```python

        self.history = np.zeros(self.episode)
        self.available_actions = np.zeros((4, 12 , 4))
        for position, action in position_action_iterator():
            position_ = position + ACTION_TO_DIRECTION[action]
            if check_available_position(position_):
                self.available_actions[position[0], position[1], action] = 1

    def start(self):
        # alphas = [0.1, 0.5, 0.9]
        alphas = [0.9]
        for self.alpha in alphas:
            for _ in tqdm(range(self.average_times)):
                self.history += (self.solve(policy='sarsa') - self.history) / (_+1)
            self.solve(policy='sarsa', verbose=True)
            self.plot_rewards('sarsa')
            self.history = np.zeros(self.episode)

            for _ in tqdm(range(self.average_times)):
                self.history += (self.solve(policy='q_learning') - self.history) / (_+1)
            self.plot_rewards('Q\ Learning')
            self.solve(policy='q_learning', verbose=True)
            self.history = np.zeros(self.episode)

        if self.is_plot:
            plot_savefig(f"$\\epsilon={self.epsilon}$",
                         f"epsilon={self.epsilon},alphas={alphas},gamma={self.gamma},av

    def solve(self, policy, verbose=False):
        history = []
        policies = ['sarsa', 'q_learning']
        assert(policy in policies)

        q = np.random.normal(loc=0, size=(4, 12, 4))
        for action in range(4):
            q[3, 11, action] = 0
        for position, action in position_action_iterator():
            x, y = position
            if not self.available_actions[x, y, action]:
                q[x, y, action] = -np.inf

        def epsilon_choose(state):
            actions = np.argwhere(self.available_actions[state[0], state[1]] > 0).reshape(-1)
            randnum = np.random.rand(1)[0]
            if randnum <= self.epsilon:
                random_action = np.random.randint(0, actions.shape[0])
                return actions[random_action]
            else:
                return np.argmax(q[state[0], state[1]])
```

```python

        for _ in range(self.episode):
            env = WalkingEnvironment()
            state = env.reset()
            terminal = False
            action = epsilon_choose(state) if policy == 'sarsa' else None
            total_reward = 0
            total_step = 0
            while not terminal:
                total_step += 1
                action = action if policy == 'sarsa' else epsilon_choose(state)
                reward, state_, terminal = env.step(action)

                if policy == 'sarsa':
                    action_ = epsilon_choose(state_)
                else:
                    action_ = np.argmax(q[state_[0], state_[1]])
                error = reward + self.gamma * q[state_[0], state_[1], action_] - q[state[0], state[1], action]
                q[state[0], state[1], action] += self.alpha * error

                state, action = state_, action_ if policy == 'sarsa' else action
                total_reward = reward + total_reward
            history.append(total_reward)

        if verbose:
            print(f"{policy}'s best policy:")
            state = env.reset()
            step_history = [tuple(state)]
            terminal = False
            while not terminal:
                action = np.argmax(q[state[0], state[1]])
                reward, state, terminal = env.step(action)
                step_history.append(tuple(state))
            print(step_history)
        return np.array(history)

    def plot_rewards(self, name):
        # plt.plot(self.history, label=f'${name}, \\epsilon={self.epsilon}, \\alpha={self.alpha:.1f}, \\gamma={self.gamma}$ ')
        plt.plot(self.history, label=f'${name}, \\epsilon={self.epsilon}$', ls='-' if name == 'sarsa' else '--')

if __name__ == '__main__':
    plt.figure(figsize=(10, 8))
    agent = Agent(is_plot=False)
    epsilons = [0, 0.01, 0.1]
    # agent.average_times = 10
    for epsilon in epsilons:
        agent.epsilon = epsilon
        agent.start()
    plot_savefig(f"$sarsa\\quad v.s.\\quad Q\ Learning$\n$\\alpha={agent.alpha:.1f}, \\gamma={agent.gamma}$",
```

```
                        ↪ f"epsilon={epsilons},alphas={agent.alpha},gamma={agent.gamma},average=

if __name__ == '__main__0':
    x, y, z = np.array([-np.inf, -np.inf, -1e9])
    print(x, y, z)
```