强化学习期末作业

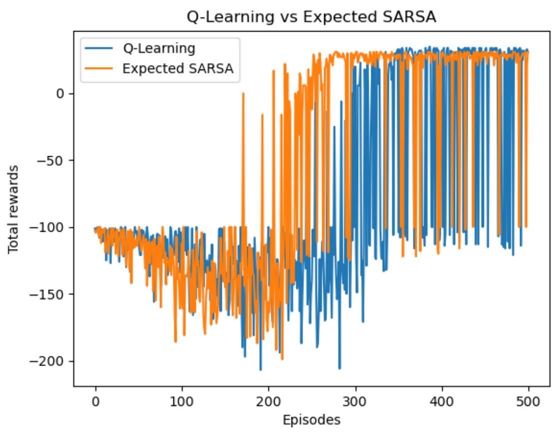
赵天钧

学号：23210180134

2025 年 2 月 19 日

本期末报告解答第一题的悬崖寻路问题，分别设计了 Q 学习方法和期望 SARSA 方法，比较了其在线性能，并尝试对探索度进行优化。另外还附上了一个基于 Q 学习的 UCB 函数优化，对探索度的平衡进一步改进 (即第二题的内容的部分)。

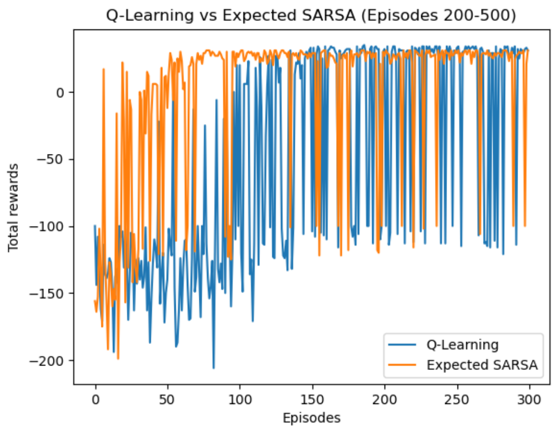
下面是测试结果：



Q 学习与期望 SARSA 性能对比

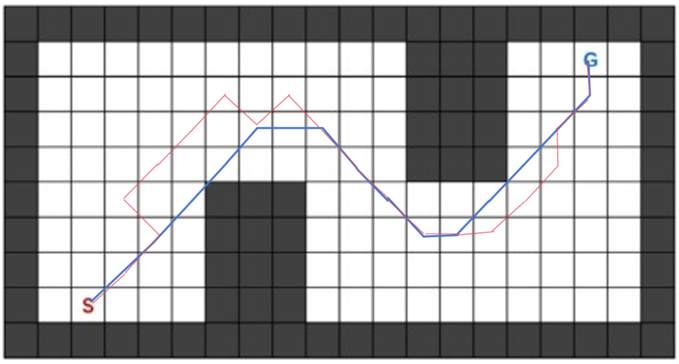
从图 1 可以看出，两种方法起初都在探索，并都以落下悬崖为止。由于期望 SARSA 探索性更强，而普通行路有惩罚，导致得分更低。但这也让期望 SARSA 更早寻得终点，并很快收敛到最优行路。而 Q 学习则由于更加贪

心，导致在探索到终点后选择更危险的路径，始终有较大概率掉下悬崖。期望 SARSA 学到更长但更安全的路径，于是在图的上侧可以看到，SARSA 的得分略小 (因为路径长也有惩罚) 但更稳定。图 2 更鲜明的反应了靠后阶段两种方法性能的对比。



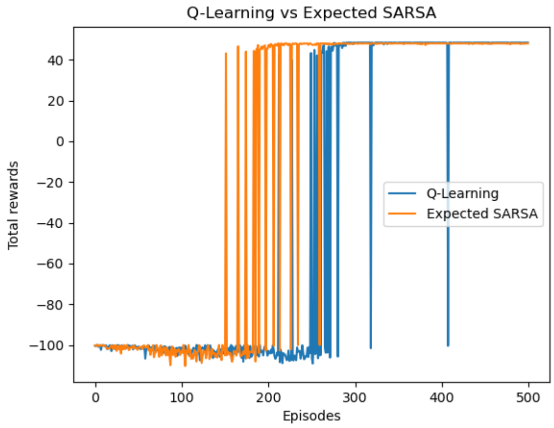
Q 学习与期望 SARSA 性能对比，靠后阶段

下图展示两种方法学到的最终路线



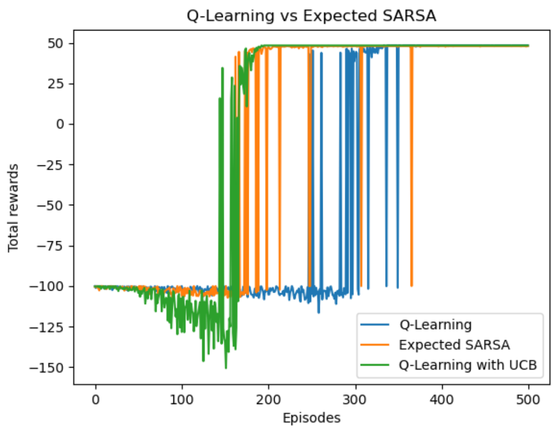
Q 学习与期望 SARSA 结果对比

适当减小路径惩罚，并让 ϵ 逐步减小，两种方法最终都收敛到最优策略



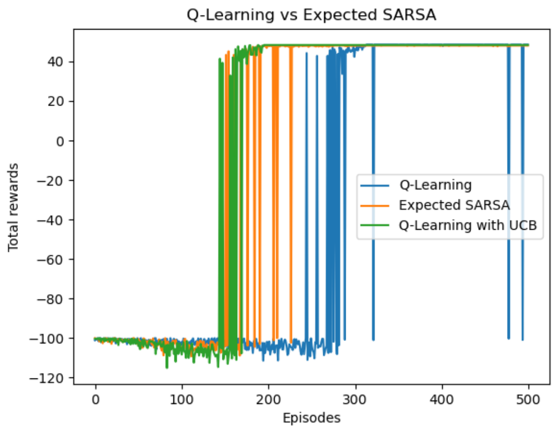
Q 学习与期望 SARSA 结果对比，epsilon 逐步减小

与 UCB 优化后的 Q 学习对比可以看到，相比于直接要求 ϵ 线性减小，UCB 函数更严格地达到了次线性总遗憾，前期的探索更强，后期也稳定表现为最优策略。



减弱探索的 Q 学习与期望 SARSA 和 UCB 优化的 Q 学习对比

其中关键系数 c 只要大于零 (不退化为原始 Q 学习) 就对性能有很大提升



c=0.1 的性能对比

源代码如下

```
import numpy as np
import matplotlib.pyplot as plt

# 设置悬崖寻路问题的环境
class CliffWalkingEnv:
def __init__(self):
self.height = 10
self.width = 20
self.start_state = (1, 2)
self.goal_state = (8, 17)
self.actions = [(0, 1), (1, 0), (0, -1), (-1, 0), (1, 1), (-1, -1), (1, -1), (-1, 1
# 个行动方向8
self.state = self.start_state
self.done = False  # 判定单个回合是否结束
self.cliff_areas = [(i, j) for i in range(0, 5) for j in range(6, 9)] + \
[(i, j) for i in range(5, 10) for j in range(12, 15)]   # 额外悬
崖设置

def reset(self):
self.state = self.start_state
self.done = False
return self.state
```

```
def step(self, action):
if self.done:
raise Exception("Episode is done")

next_state = (self.state[0] + action[0], self.state[1] + action[1])
# 一般寻路行进的状态更新

if next_state == self.goal_state:
reward = 50
next_state = self.start_state
self.done = True    # 到达终点给予奖励，并视为完成一次寻路
elif (next_state[0] == 0 or next_state[0] == 9 or
next_state[1] == 0 or next_state[1] == 19 or
next_state in self.cliff_areas):
reward = -100
next_state = self.start_state
self.done = True  # 到达悬崖给予负的奖励，也视为完成一次寻路
else:
reward = -0.1 # 一般移动也给予一个惩罚，以便更快收敛和寻得更短路径否则
智能体在初始阶段容易陷入死循环来规避悬崖,

self.state = next_state
return next_state, reward, self.done



def q_learning(env, episodes, alpha, gamma, epsilon_start, epsilon_end): # 设
计学习算法Q
q_table = np.zeros((env.height, env.width, len(env.actions)))
total_rewards = []
paths = []

for episode in range(episodes):
epsilon = max(epsilon_start - (epsilon_start - epsilon_end) * (episode / episodes),
心决定了智能体的探索程度，我们希望探索性逐渐减小，逐步从探索为主变为利用为主
导epsilon
state = env.reset()
done = False
```

```
total_reward = 0  # 初始化值函数，总奖励和寻路完成指标q

path = [state]

while not done:
if np.random.rand() < epsilon:
action = np.random.choice(len(env.actions))
else:
action = np.argmax(q_table[state])  # 基于值函数和贪心方法寻得此步
的动作决策qepsilon

next_state, reward, done = env.step(env.actions[action])  # 调用
环境设置获知决策后状态和单步奖励
next_max = np.max(q_table[next_state])

q_table[state + (action,)] = q_table[state + (action,)] + alpha * (
reward + gamma * next_max − q_table[state + (action,)])  # 更新
值函数，其中是学习率，是遗忘率qalphagamma

# 状态更新，累加总奖励和当前状态
state = next_state
total_reward += reward
path.append(state)

# 储存每个的总奖励和路径episode
total_rewards.append(total_reward)
paths.append(path)

if (episode + 1) % 100 == 0:
print(f"Q−Learning Episode {episode + 1}: Path = {path}")
# 这是测试时观察不同方法在学习过程中的表现，每个100输出一次结果，可删
去episode

# 输出每个总奖励以供绘图episode
return total_rewards


def expected_sarsa(env, episodes, alpha, gamma, epsilon_start, epsilon_end): # 设
```

计期望的算法，与学习方法很相似，重复的注释不再赘述SARSAQ

```
q_table = np.zeros((env.height, env.width, len(env.actions)))
total_rewards = []
paths = []

for episode in range(episodes):
epsilon = max(epsilon_start - (epsilon_start - epsilon_end) * (episode / episodes),
state = env.reset()
done = False
total_reward = 0
path = [state]

while not done:
if np.random.rand() < epsilon:
action = np.random.choice(len(env.actions))
else:
action = np.argmax(q_table[state])

next_state, reward, done = env.step(env.actions[action])

policy_prob = np.ones(len(env.actions)) * epsilon / len(env.actions)
best_action = np.argmax(q_table[next_state])
policy_prob[best_action] += (1.0 - epsilon)

expected_q = np.dot(q_table[next_state], policy_prob) # 这一段主
```

要是为了加速运算，因为期望涉及期望计算，比的代价大，也是与学习方法的根本不同SARSAQQ

```
q_table[state + (action,)] = q_table[state + (action,)] + alpha * (
reward + gamma * expected_q - q_table[state + (action,)])

state = next_state
total_reward += reward
path.append(state)

total_rewards.append(total_reward)
paths.append(path)
```

```
if (episode + 1) % 100 == 0:
print(f"Expected SARSA Episode {episode + 1}: Path = {path}")
return total_rewards


def q_learning_with_ucb(env, episodes, alpha, gamma, c): # 额外
设计了一个基于学习和方法的改进QUCB
q_table = np.zeros((env.height, env.width, len(env.actions)))
n_table = np.zeros((env.height, env.width, len(env.actions))) # 方
法的探索体现在对以往动作的记忆和创新，因此额外需要一个表格储存动作计数，稍微
牺牲一下空间UCB
total_rewards = []
paths = []

for episode in range(episodes):
state = env.reset()
done = False
total_reward = 0
path = [state]

while not done:
# 函数综合了探索与利用，因此不再需要，而是直接贪心最大化本身UCBepsilonucb
s_idx = (state[0], state[1])
total_t = np.sum(n_table[s_idx]) + 1
ucb_values = q_table[s_idx] + c * np.sqrt(np.log(total_t) / (n_table[s_idx] + 1)) #
数由值函数和置信度评分共同组成UCBq
action_idx = np.argmax(ucb_values)
action = env.actions[action_idx]

next_state, reward, done = env.step(action)

ns_idx = (next_state[0], next_state[1])
best_next_action = np.argmax(q_table[ns_idx])
q_table[s_idx + (action_idx,)] += alpha * (
reward + gamma * q_table[ns_idx + (best_next_action,)] - q_table[s_idx + (action_idx

# 动作计数
n_table[s_idx + (action_idx,)] += 1
```

```
# 状态更新
state = next_state
total_reward += reward
path.append(state)

total_rewards.append(total_reward)
paths.append(path)
return total_rewards


# 设置环境和参数
env = CliffWalkingEnv()
episodes = 500
alpha = 0.5
gamma = 0.9
epsilon_start = 0.2
epsilon_min = 0.005
epsilon_end = -0.1 # 这是为了让逐步减小的设置，设置相同的始末则退化为
不变的情形epsilon
c = 0.1 # 这是方法的置信度系数，可以调整以获得更优的组合UCB

# 运行算法获取各自在线性能
rewards_q_learning = q_learning(env, episodes, alpha, gamma, epsilon_start, epsilon_
rewards_expected_sarsa = expected_sarsa(env, episodes, alpha, gamma, epsilon_start,
rewards_q_learning_ucb = q_learning_with_ucb(env, episodes, alpha, gamma, c)
#for t in range(0, 5):
#     c = t/40
#     rewards_q_learning_ucb = q_learning_with_ucb(env, episodes, alpha, gamma, c)

#     plt.plot(rewards_q_learning_ucb)
# 绘制图表进行比较
plt.plot(rewards_q_learning, label='Q-Learning')
plt.plot(rewards_expected_sarsa, label='Expected SARSA')
plt.plot(rewards_q_learning_ucb, label='Q-Learning with UCB')
plt.xlabel('Episodes')
plt.ylabel('Total rewards')
plt.title('Q-Learning vs Expected SARSA') # 可以再加一个 vs Q with UCB
```

```
plt.legend()
plt.show()
# 额外观察靠后的，较稳定的学习成果
#plt.plot(rewards_q_learning[200:], label='Q-Learning')
#plt.plot(rewards_expected_sarsa[200:], label='Expected SARSA')
#plt.plot(rewards_q_learning_ucb[200:], label='Q-Learning with UCB')
#plt.xlabel('Episodes')
#plt.ylabel('Total rewards')
#plt.title('Q-Learning vs Expected SARSA (Episodes 200-500)')
#plt.legend()
#plt.show()
```