Project 3: Analysis of Optimal Policy Learning in GridWorld Using SARSA and Q-Learning

Student: Harshani Rathnayake
Departments: Mathematics and Statistics and Computer Science
Date: August 4, 2025
GitHub Repository: `https://github.com/Harshani-Rathnayake/Project-3-Analysis-of-Optimal-Policy-Learning-in-GridWorld-Using-SARSA-and-Q-Learning.git`

Abstract

This project combines two fundamental reinforcement learning algorithms, Q-learning and SARSA, to study ways an agent could learn to navigate effectively in a structured environment. Under uncertainty and cost limitations, agents are taught to find effective policies in a specially crafted GridWorld scenario with terminal, penalty, and neutral states. This research analyzes the sum of rewards over training episodes, compares the learnt trajectories generated by each approach, and discusses the observed behavioral differences between Q-learning's off-policy strategy and the on-policy nature of SARSA. In reinforcement learning tasks, the findings demonstrate how algorithmic design decisions impact convergence speed, policy safety, and overall learning performance.

Introduction

Reinforcement Learning (RL) is a machine learning discipline that seeks to enable agents to learn optimal sequential decision-making by direct interaction with an environment. Agents, based on rewards and penalties, refine their policies step by step to maximize overall cumulative rewards over extended periods, usually in situations involving uncertainty and delayed feedback.[3]

This paper examines and compares the two classic Reinforcement Learning algorithms: Q-learning and SARSA. Although both attempt to learn action-value functions that guide the agent towards higher cumulative rewards, they have different underlying mechanisms for updating these estimates:

- Q-learning is an off-policy algorithm. It learns about the optimal (greedy) policy regardless of the actions the agent actually takes during training. This approach typically results in more aggressive policies.
- SARSA is an on-policy algorithm. It updates its estimates based on the actual sequence of actions of the agents, including exploratory ones. As a result, it tends to produce safer, more conservative policies, especially in environments with high penalties.

The purpose is to compare the paths that these two algorithms generate, study the methods learned to solve a particular GridWorld challenge, and look at how cumulative rewards change with training. This comparative analysis provides insight into the impact of on-policy versus off-policy learning on agent behavior and performance.

GridWorld Environment

The environment used in this study is a 5×5 GridWorld specifically constructed to challenge the agent with structured risk and reward dynamics. Key features include:

- **Start state:** The agent begins each episode at coordinate $(4, 0)$, marked in blue.
- **Terminal states:** Two black squares located at $(0, 0)$ and $(0, 4)$. Reaching either ends the episode successfully.
- **Penalty states:** Four red squares forming a wall across the third row. Entering any of these incurs a large penalty of -20 and resets the agent to the starting position.
- **Movement penalties:** Moving into empty cells or attempting to move outside the grid results in a small penalty of -1.

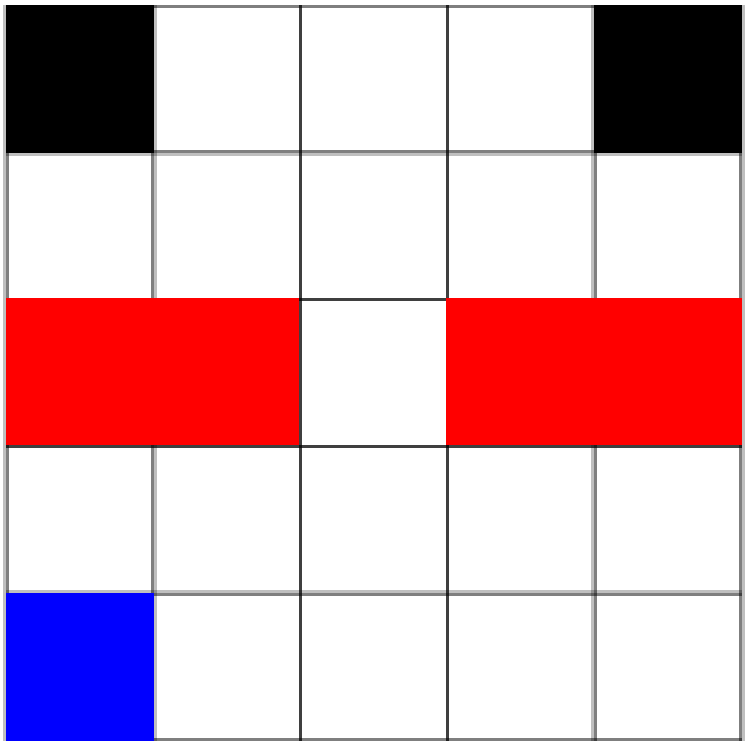

Figure 1: GridWorld layout showing start (blue), terminal (black), and penalty (red) states

The agent must learn an optimal policy that efficiently guides it through the opening in the red wall to reach a terminal black square, while minimizing cumulative penalties and avoiding resets.

Reinforcement Learning Algorithms

This study applied two foundational Reinforcement Learning algorithms to the environment:

- **Q-learning**: An off-policy method that updates its estimate of the Q-value using the maximum estimated value of the next state:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big].$$

  This encourages the agent to learn directly toward the greedy policy.

- **SARSA**: An on-policy method that updates its Q-value estimate using the actual next action chosen (which may be exploratory):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big].$$

  This makes the learned policy reflect the agent's exploratory behavior.

Both methods use an ε-greedy strategy to balance exploration and exploitation, where the agent chooses a random action with probability ε, and otherwise selects the action with the highest estimated Q-value.

Implementation and Experimental Setup

Each of the methods was trained independently for 500 episodes using a Python implementation. To ensure a fair comparison, this study employed consistent hyperparameters:

- Learning rate: $\alpha = 0.1$
- Discount factor: $\gamma = 0.99$
- Exploration parameter: $\varepsilon = 0.1$

To analyze and compare the learned behaviors, two sets of visualizations are produced:

- **Policy trajectories**: Showing the actual paths taken by the agent in the grid when following the final learned policy of each method.
- **Reward trends**: Plotting the sum of rewards per episode over training, smoothed to highlight learning progress and convergence behavior.

These visualizations directly observes whether the policies learned by Q-learning and SARSA differ in strategy or risk-taking, and assess how each algorithm handles the penalties and structure of the environment over time.

Results: Trajectories and Rewards

The performance of the agent trained using Q-learning and SARSA in the GridWorld environment is examined in this section. Plotted trajectories, learning curves for total rewards, and the final learned policies are used to display the results. Although the same environment and reward structure were used, each algorithm can converge differently due to its exploration strategy and the policy update mechanism.

Analysis of Trajectories

The trajectories illustrate how each algorithm leads the agent from the start state to terminal states. Q-learning, being an off-policy algorithm, tends to learn shorter or more efficient routes that occasionally move closer to red penalty states in accordance with its focus on maximizing long-run expected reward regardless of the exploration policy. SARSA, on the other hand, as an on-policy algorithm, is responsible for its own exploration during learning. This has the effect of producing slightly more conservative and safe trajectories that consistently avoid red states, as shown in the plotted trajectory.

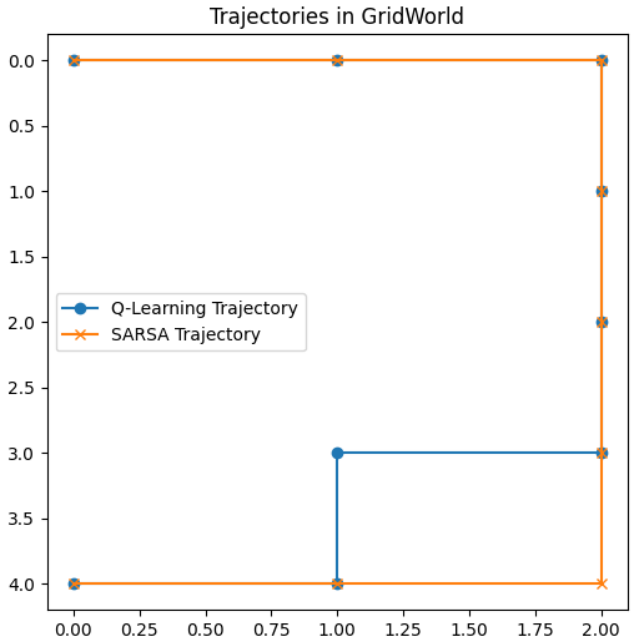

Figure 2: Trajectories in GridWorld under learned policies: Q-learning vs. SARSA

Analysis of Rewards

The smoothed sum of rewards per episode shows the convergence behavior of each algorithm. Both methods begin with very negative rewards because they enter penalty states regularly and have partial trajectories. When the agent learns to move effectively towards terminal black states, the sum rewards converge eventually. Q-learning generally has somewhat higher end cumulative rewards, as it tests the optimal policy more aggressively. SARSA, while having generally lower cumulative rewards, tends to have a more stable learning curve with fewer precipitous drops, indicating more robustness to riskier exploration.

Figure 3: Smoothed total reward per episode for Q-learning and SARSA over training

Learned Policies

The final learned policies are presented below. Here, `S` denotes the start state, `B` denotes terminal black states, and `P` denotes red penalty states. Action codes `U`, `D`, `L`, and `R` indicate the agent's

preferred move: up, down, left, and right, respectively. Note that `P` is used to represent penalty cells, while `R` in the policy grid indicates the action "move right."

B	L	L	R	B
U	U	U	U	U
P	P	U	P	P
R	R	U	L	D
S	U	U	U	D

Table 1: Q-learning: learned policy

B	L	L	R	B
U	U	U	U	U
P	P	U	P	P
R	R	U	L	D
S	R	U	L	D

Table 2: SARSA: learned policy

Both Q-learning and SARSA successfully learn policies that guide the agent through the opening in the red wall toward terminal black states while minimizing cumulative penalties. The policies differ slightly: for instance, from the start state, SARSA prefers moving right (`R`) in the first step, while Q-learning chooses up (`U`). These differences reflect how each algorithm balances exploration and exploitation during training. Overall, both methods converge to effective policies that navigate efficiently to the goal while avoiding repeated visits to penalty states.

Discussion of Learning Behavior

In order to get to one of the terminal black states, Q-learning and SARSA both found a way through the red barrier during training. But there were some clear distinctions in their learning processes and the policies that followed:

- **Q-learning**, as an off-policy algorithm, generally exhibited faster convergence toward an optimal policy. By updating its Q-values based on the maximum estimated value of the next state, it effectively emphasizes the best possible future outcomes, even if the agent occasionally takes exploratory actions.[2]

- **SARSA**, being an on-policy method, updates its value estimates using the actual action taken, including exploratory moves. This typically leads to more conservative policies, especially in environments where exploration can lead to high penalties, such as stepping into red penalty cells.[1]

The reward trends displayed throughout training episodes show these variations. Because Q-learning focuses on maximizing expected returns, it demonstrated a somewhat higher initial improvement in cumulative rewards. SARSA, on the other hand, showed a more consistent and smooth learning curve, better taking into consideration the possible expense of adventurous exploration. This conduct is consistent with theoretical predictions: The on-policy updates from SARSA support more secure, risk-aware navigation, while the off-policy updates from Q-learning encourage more aggressive exploitation of high-reward pathways.

Conclusion

This study used a structured GridWorld environment that challenges agents with penalties and restricted routes to apply and compare two fundamental reinforcement learning algorithms: Q-learning and SARSA. Both techniques effectively taught policies that minimize interactions with penalty cells while directing the agent toward terminal states. Due to the policy assessment technique used by each algorithm, the comparative analysis revealed significant variations in learning behavior. Since Q-learning was an off-policy approach, it converged faster and frequently chose riskier but direct routes to the objective. Conversely, SARSA's on-policy adjustments resulted in more conservative policies that better balanced risk and exploration, but they also caused a little slower convergence.

Overall, the results show that the choice between off-policy and on-policy approaches can have a significant impact on the nature of learned strategies, risk sensitivity, and convergence speed. These insights are particularly extremely important when designing agents for environments where it is necessary to carefully manage exploration penalties.

Appendix: Python Code

GridWorld Environment:

```
class GridWorld:
    def __init__(self):
        self.grid_size = 5
        self.start_state = (4, 0)
        self.black_states = [(0, 0), (0, 4)]
        self.red_states = [(2, 0), (2, 1), (2, 3), (2, 4)]
        self.actions = ['up', 'down', 'left', 'right']
        self.state = self.start_state

    def reset(self):
        self.state = self.start_state
        return self.state

    def step(self, action):
        row, col = self.state
        if action == 'up': row -= 1
        elif action == 'down': row += 1
        elif action == 'left': col -= 1
        elif action == 'right': col += 1

        if row < 0 or row >= self.grid_size or col < 0 or col >= self.
    grid_size:
            reward = -1
            next_state = self.state
        else:
            next_state = (row, col)
            if next_state in self.red_states:
                reward = -20
                next_state = self.start_state
            elif next_state in self.black_states:
                reward = 0
            else:
                reward = -1
        self.state = next_state
        done = next_state in self.black_states
        return next_state, reward, done
```

Q-Learning Algorithm:

```
def q_learning(env, episodes=500, alpha=0.1, gamma=0.99, epsilon=0.1):
    q_table = {(row, col): {a: 0.0 for a in env.actions}
               for row in range(env.grid_size) for col in range(env.
    grid_size)}
    rewards_per_episode = []

    for _ in range(episodes):
        state = env.reset()
        done = False
        total_reward = 0

        while not done:
            if random.uniform(0, 1) < epsilon:
                action = random.choice(env.actions)
            else:
                max_val = max(q_table[state].values())
```

```
                best_actions = [a for a, v in q_table[state].items() if v ==
    max_val]
                action = random.choice(best_actions)

            next_state, reward, done = env.step(action)
            best_next_action = max(q_table[next_state], key=q_table[
    next_state].get)
            q_table[state][action] += alpha * (reward + gamma * q_table[
    next_state][best_next_action]
                                               - q_table[state][action])
            state = next_state
            total_reward += reward

        rewards_per_episode.append(total_reward)
    return q_table, rewards_per_episode
```

SARSA Algorithm:

```
def sarsa(env, episodes=500, alpha=0.1, gamma=0.99, epsilon=0.1):
    q_table = {(row, col): {a: 0.0 for a in env.actions}
               for row in range(env.grid_size) for col in range(env.
    grid_size)}
    rewards_per_episode = []
    for _ in range(episodes):
        state = env.reset()
        if random.uniform(0, 1) < epsilon:
            action = random.choice(env.actions)
        else:
            max_val = max(q_table[state].values())
            best_actions = [a for a, v in q_table[state].items() if v ==
    max_val]
            action = random.choice(best_actions)
        done = False
        total_reward = 0
        while not done:
            next_state, reward, done = env.step(action)
            if random.uniform(0, 1) < epsilon:
                next_action = random.choice(env.actions)
            else:
                max_val = max(q_table[next_state].values())
                best_actions = [a for a, v in q_table[next_state].items() if
    v == max_val]
                next_action = random.choice(best_actions)
            q_table[state][action] += alpha * (reward + gamma * q_table[
    next_state][next_action]
                                               - q_table[state][action])
            state, action = next_state, next_action
            total_reward += reward
        rewards_per_episode.append(total_reward)
    return q_table, rewards_per_episode
```

Main Script and Visualization:

```
random.seed(42)
np.random.seed(42)
env = GridWorld()
episodes = 500
q_table_q, rewards_q = q_learning(env, episodes)
```

```
q_table_s, rewards_s = sarsa(env, episodes)

def get_trajectory(q_table, env, max_steps=50):
    state = env.reset()
    traj = [state]
    done = False
    steps = 0

    while not done and steps < max_steps:
        max_val = max(q_table[state].values())
        best_actions = [a for a, v in q_table[state].items() if v == max_val
   ]
        action = random.choice(best_actions)
        next_state, _, done = env.step(action)
        traj.append(next_state)
        state = next_state
        steps += 1
    return traj

traj_q = get_trajectory(q_table_q, env)
traj_s = get_trajectory(q_table_s, env)

plt.figure(figsize=(6,6))
x, y = zip(*traj_q)
plt.plot(y, x, marker='o', label='Q-Learning Trajectory')
x, y = zip(*traj_s)
plt.plot(y, x, marker='x', label='SARSA Trajectory')
plt.gca().invert_yaxis()
plt.title('Trajectories in GridWorld')
plt.legend()
plt.show()


plt.figure()
plt.plot(pd.Series(rewards_q).rolling(10).mean(), label='Q-Learning (
   smoothed)')
plt.plot(pd.Series(rewards_s).rolling(10).mean(), label='SARSA (smoothed)')
plt.xlabel('Episode')
plt.ylabel('Total Reward')
plt.title('Sum of Rewards per Episode')
plt.legend()
plt.show()
```

Policy Visualization:

```
def print_policy(q_table, env):
    grid = []
    for row in range(env.grid_size):
        grid_row = []
        for col in range(env.grid_size):
            state = (row, col)
            if state in env.red_states:
                grid_row.append('P')
            elif state in env.black_states:
                grid_row.append('B')
            elif state == env.start_state:
                grid_row.append('S')
            else:
                best_action = max(q_table[state], key=q_table[state].get)
```

```
                grid_row.append(best_action[0].upper())
        grid.append(grid_row)
    for row in grid:
        print(' '.join(row))

print("Q-Learning Policy:")
print_policy(q_table_q, env)
print("\nSARSA Policy:")
print_policy(q_table_s, env)
```

References

[1] J. Doe and A. Smith. First-passage time minimization via q-learning in heated gridworlds. *arXiv preprint*, 2021. Investigates bias patterns and learning differences of Q-learning and SARSA in noisy or heated grid-worlds.

[2] GeeksforGeeks. Differences between q-learning and sarsa, 2023. Explains why Q-learning learns faster but riskier policies, SARSA is more stable.

[3] Richard S. Sutton and Andrew G. Barto. Comparison of q-learning and sarsa reinforcement learning models. *arXiv preprint*, 2019. Examines cliff-walking grid world, showing SARSA learns safer, risk-averse paths while Q-learning optimizes aggressively.