Navigating a 10x10 Grid

By Karingattil Sagar Thomas , Vitthal Vinod Tiwary

2022A7PS0156H , 2022A3PS0553H

Problem Statement

We have chosen a problem involving navigating a **10x10 grid**, starting from the **top-left corner at position (0,0)** with the objective of reaching the **bottom-right corner at position (9,9)**. The grid contains special cells with distinct properties:

- **Cliffs (Kill States):**
  - These cells act as **cliff cells** that result in an immediate penalty of **-20** and teleport the player back to the starting position **(0,0)**.
  - The cliffs are located at the following positions:

```
(0,4), (0,5), (0,6), (0,7), (0,8),
(1,4), (1,5), (1,6), (1,7), (1,8),
(3,1), (4,1), (5,1), (6,1), (7,1),
(3,2), (4,2), (5,2), (6,2), (7,2),
(6,4), (6,5), (6,6), (6,7), (6,8),
(7,4), (7,5), (7,6), (7,7), (7,8)
```

- **Goal State:**
  - The **goal** is to reach the cell at **(9,9)**.
  - Reaching this state completes the game with a **reward of 0**.

- **Regular Cells:**
  - Moving to any other cell (except the goal) results in a **reward of -1** for each step taken.

Objective

The player starts at the **initial position (0,0)** and must find a path to the **goal state at (9,9)** while avoiding the cliffs. The objective is to **minimize the total penalty** incurred during the journey and reach the goal with the fewest steps possible.

Grid Layout

R\C	0	1	2	3	4	5	6	7	8	9
0	S				X	X	X	X	X	
1										
2										X
3	X		X							X
4	X		X							X
5	X		X							X
6	X		X		X	X	X			X
7	X		X			X				
8						X				
9										T

Table 1: 10x10 Grid Layout

Q-Learning

Q-Learning is an **off-policy reinforcement learning** algorithm used to learn the optimal action-value function, which helps the agent make the best decisions in a given environment. Unlike on-policy methods, Q-Learning updates the policy independently of the actions taken, enabling it to learn from simulated or random experiences.

The Q-value function, $Q(s, a)$, estimates the expected cumulative reward of taking action a in state s and following the optimal policy thereafter. The update rule for Q-Learning is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

where:

- $Q(s_t, a_t)$ is the current estimate of the Q-value for state s_t and action a_t.
- α is the **learning rate**, which determines how much new information overrides the old value. Typical values range between 0 and 1.
- R_{t+1} is the reward received after taking action a_t in state s_t.
- γ is the **discount factor**, controlling the importance of future rewards (here, $\gamma = 0.8$).
- $\max_{a'} Q(s_{t+1}, a')$ is the estimated optimal future reward for the next state s_{t+1}.

Epsilon-Greedy Policy

To balance exploration and exploitation, the agent follows an **epsilon-greedy policy**. This policy ensures that the agent explores new actions with some probability ϵ while exploiting the best-known action most of the time.

The epsilon-greedy policy is defined as:

$$\pi(s, a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{otherwise} \end{cases}$$

where:

- $\pi(s, a)$ is the probability of taking action a in state s under the policy.
- $Q(s, a')$ is the Q-value for action a' in state s.
- $\epsilon = 0.2$ is the exploration parameter, which ensures the agent takes random actions 20% of the time.
- $|\mathcal{A}(s)|$ is the number of available actions in state s.

Learning Process

The Q-Learning process consists of the following steps:

1. Initialize $Q(s, a)$ arbitrarily for all state-action pairs.
2. For each episode, start at the initial state and select actions using the epsilon-greedy policy.
3. Update the Q-values using the Q-Learning update rule.
4. Continue until convergence or for a fixed number of episodes.

With Q-Learning, the goal is to learn the optimal Q-values, which allows the agent to derive the optimal policy for navigating the environment efficiently.

Double, Triple, and Quadruple Q-Learning

While standard Q-Learning can suffer from overestimation bias due to the use of the same Q-values for both action selection and value estimation, Double, Triple, and Quadruple Q-Learning introduce multiple Q-value estimators to address this bias and improve learning stability.

Double Q-Learning‘

In **Double Q-Learning**, two separate Q-value tables, Q_1 and Q_2, are maintained. These tables alternate during updates, reducing the bias caused by overestimation. The update rule for Double Q-Learning is:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha \left(R_{t+1} + \gamma Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1}, a')) - Q_1(s_t, a_t) \right)$$

or vice versa, swapping Q_1 and Q_2 during updates.
Key Differences:

- Two Q-tables (Q_1 and Q_2) are maintained to reduce overestimation bias.
- During each update, one Q-table selects the action, while the other estimates the value.

Triple Q-Learning

Triple Q-Learning extends the idea further by introducing a third Q-value estimator, Q_3. This improves stability in environments with high variability.

The update for one of the tables, say Q_1, is:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha \left(R_{t+1} + \gamma Q_i(s_{t+1}, \arg\max_{a'} Q_j(s_{t+1}, a')) - Q_1(s_t, a_t) \right)$$

where $i, j \in \{2, 3\}$, and $i \neq j$.
Key Differences:

- Three Q-tables (Q_1, Q_2, Q_3) are used to improve stability.
- Updates use two Q-tables: one for action selection and another for value estimation.

Quadruple Q-Learning

In **Quadruple Q-Learning**, four Q-value estimators, Q_1, Q_2, Q_3, and Q_4, are maintained. This method further reduces the risk of overestimation and increases robustness in complex environments.

The update rule for one of the Q-tables, say Q_1, is:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha \left(R_{t+1} + \gamma Q_i(s_{t+1}, \arg\max_{a'} Q_j(s_{t+1}, a')) - Q_1(s_t, a_t) \right)$$

where $i, j \in \{2, 3, 4\}$, and $i \neq j$.
Key Differences:

- Four Q-tables (Q_1, Q_2, Q_3, Q_4) are maintained.
- Similar to Triple Q-Learning, updates use two of the remaining Q-tables for action selection and value estimation.
- This method offers improved robustness for environments with high uncertainty or noise.

Summary of Differences from Simple Q-Learning

- **Multiple Q-Tables:** Unlike simple Q-Learning, which uses a single Q-table, these methods introduce multiple Q-tables to reduce overestimation bias.
- **Action-Value Decoupling:** Action selection and value estimation are decoupled by using different Q-tables.
- **Improved Stability:** Triple and Quadruple Q-Learning offer greater stability and robustness, particularly in stochastic environments.

1 Summary of Code Functionality

1.1 Initialization

The code defines a 10×10 grid with different types of states:

- **Goal State:** Located at $(9, 9)$, with a reward of 0.
- **Initial State:** Set at $(0, 0)$.
- **Kill States:** Specific grid positions where the agent resets to the initial state and receives a reward of -20.
- **Other States:** All other states have a reward of -1.

The code also defines a discount factor $\gamma = 0.8$, learning rate $\alpha = 0.1$, and an ϵ-greedy policy with decay.

1.2 Functions

- `find_first_occurrence_index(episode, target_tuple)`: Finds the first occurrence of a specific state-action pair in an episode.
- `sum_rewards_from_index(episode, start_index)`: Calculates the total discounted reward from a given starting point.
- `get_next_state(state, action)`: Determines the next state based on the current state and action taken.

- `generate_episode(policy, epsilon, counting)`: Generates an episode using a given policy, handling state transitions and rewards. If the agent exceeds a threshold number of steps, the episode is restarted.

1.3 Q-Learning Algorithm

- Initializes the action-value function Q and ϵ-greedy policy.
- Updates the policy to select actions that maximize Q values, with exploration controlled by ϵ.
- For each episode, state-action pairs are used to update Q based on the observed rewards and the maximum value of future states.
- Decays ϵ over time to reduce exploration.

1.4 Double Q-Learning Algorithm

- Maintains two action-value functions, Q_1 and Q_2, to mitigate overestimation of state-action values.
- Updates Q_1 or Q_2 at random, using the other function to select the best next action.
- Combines Q_1 and Q_2 to create the final policy.
- Decays ϵ gradually over episodes.

1.5 Triple Q-Learning Algorithm

- Similar to double Q-learning but with three action-value functions, Q_1, Q_2, and Q_3.

1.6 Quadruple Q-Learning Algorithm

- similar to triple Q-Learning but with four action value functions

1.7 Policy and Episode Management

- The policies for each algorithm are updated based on the learned Q values to be more greedy over time.
- Episodes are generated using the current policy and stored for training purposes.

1.8 Data Collection and Visualization

- Runs experiments across multiple episodes to evaluate the performance of each policy.
- Collects metrics such as rewards, episode lengths, and Q-values.
- Visualizes the results by plotting graphs of performance metrics against the number of episodes.
- Fits best-fit lines to analyze trends and performance improvements over time.

Q-Learning Variations: Single, Double, Triple, and Quadruple Q-Learning Models

Scatter Plot with Best Fit Lines for R_Q Values

Observations:

- **Q1 (Single Q-learning):** Shows a modest increasing trend with E-values.
- **Q2 (Double Q-learning):** Demonstrates a clear upward trend, suggesting better performance over episodes.
- **Q3 (Triple Q-learning):** Shows a downward trend, possibly indicating performance degradation over time.

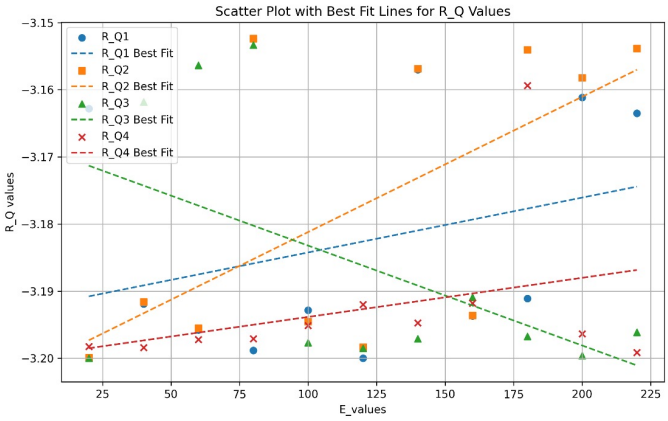

Figure 1: Rewards for different Q-learning variants

- **Q4 (Quadruple Q-learning):** Exhibits a slight positive trend but fluctuates more compared to Q1 or Q2.

Performance Analysis:

- **Double Q-learning (Q2)** performs best, showing consistent improvement without overestimation bias.
- **Single Q-learning (Q1)** remains stable but improves slower than Q2.
- **Triple Q-learning (Q3)** struggles, possibly due to the complexity of managing multiple Q-tables, leading to instability.
- **Quadruple Q-learning (Q4)** has mixed results, with both improvement and instability, reflecting the diminishing returns of adding more estimators.

Average Episode Lengths vs E-Values

Observations:

- **Q3 (Triple Q-learning):** Shows very high spikes, indicating instability and large variations in episode lengths.
- **Q1, Q2, and Q4:** Have more stable trends, with Q2 maintaining the shortest average lengths throughout.

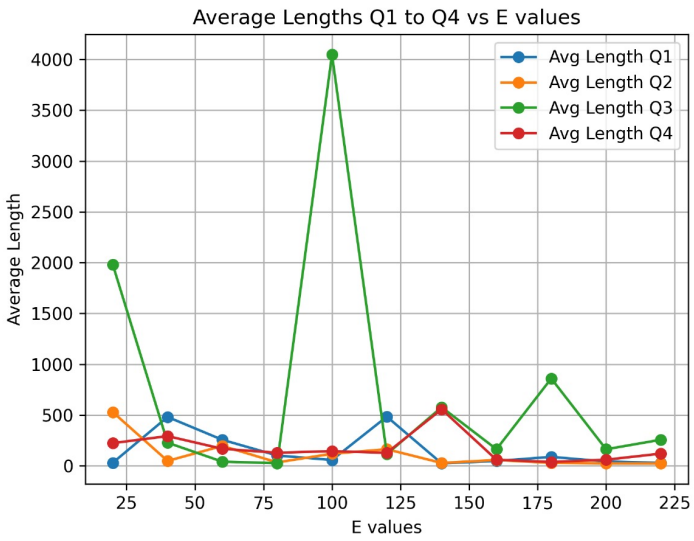

Figure 2: Average Length of variants of q at each epsiode

- **Q1 (Single Q-learning):** Demonstrates steady, though not optimal, performance.
- **Q4 (Quadruple Q-learning):** Fluctuates between Q2's low values and Q3's high spikes, showing inconsistent behavior.

Performance Insights:

- **Double Q-learning (Q2)** is the most efficient, with shorter average lengths indicating faster convergence.
- **Triple Q-learning (Q3)** struggles with stability, possibly due to the added complexity.
- **Quadruple Q-learning (Q4)** achieves some improvement but lacks consistency, suggesting increased overhead without significant benefit.

Overall Analysis and Recommendation

- **Double Q-learning (Q2)** is the optimal approach, offering a good balance of simplicity and performance. Its ability to mitigate over-

estimation bias and achieve faster convergence makes it the preferred model.

- **Single Q-learning (Q1)** is suitable for simpler tasks where performance demands are low.
- **Triple (Q3) and Quadruple (Q4) Q-learning** introduce unnecessary complexity and instability, making them less practical unless a specific need arises.

Conclusion: This detailed analysis highlights that while multiple Q-value estimators can improve performance, there is a trade-off between complexity and effectiveness. Double Q-learning strikes the best balance, making it a recommended choice for most applications.