Master 2 Analyse et Politique Économique – DS2E
Semestre 1

UE Machine Learning – Reinforcement Learning

The Cliff-Walking Problem: From Bellman’s Equation to SARSA and Q-Learning algorithms

Supervisor: Prof. Koebel Bertrand

Authors:
Vasta Francesca, Vu Billy

Academic Year: 2024/2025

Summary of Content

1. Introduction

1.1 Introduction to Reinforcement Learning and its importance

Reinforcement Learning (RL) is a branch of Machine Learning which enables an agent, or "*decision-maker*", to learn and take decisions through interactions with its environment. Unlike supervised learning, where the model is trained on labeled data with a supervisor, RL focuses on learning optimal behaviors through trial and error, receiving feedback in the form of rewards or penalties. Since the agent lacks prior knowledge of the environment, the learning process must balance *exploration* (trying new actions) and *exploitation* (choosing known successful actions). A key aspect of RL is indeed the fact that each action influences future rewards, making the decision-making process dynamic and long-term.

In RL, the ultimate objective is for the agent to develop the optimal strategy, or optimal "*policy*", that not only achieves short-term success but also maximizes long-term cumulative rewards, leading to the best possible behavior for solving the given task.

1.2 Presentation of the Cliff Walking Problem as a classic example

In this essay, we will discuss the Cliff Walking problem, which is a classic toy-example of an episodic task in reinforcement learning. Before outlining the characteristics of the Cliff Walking environment, it is useful to introduce the concepts of *episodes* and *steps*.

Episodic tasks consist of sequences of interactions between the agent and the environment, referred to as episodes. Each episode begins in an *initial state* and proceeds step by step until the agent reaches a *terminal state*, which typically represents the goal. Each individual interaction between the agent and the environment within an episode is called a step. At each step, the agent observes the current state, takes an action, and receives feedback in the form of a reward (or penalty), which then leads to a transition to a new state. The episode continues until the terminal state is reached, at which point the environment resets, and a new episode begins.

The Cliff Walking environment challenges the agent to navigate a grid world that includes a dangerous cliff. Starting from an initial position - the *initial state* - the agent's objective is to reach the goal - or *terminal state* - which is located on the opposite side of the cliff. Falling off the cliff not only incurs a significant penalty but also truncates the episode and resets the environment, returning the agent to its initial position. As previously mentioned, the overall goal of any reinforcement learning task is to navigate the environment while maximizing cumulative reward. In the context of the Cliff Walking Problem, this means training the agent to reach the terminal state while *maximizing the reward* and *minimizing the number of steps* taken in each episode.

1.3 Objectives of the essay

The Cliff Walking Problem serves as a vital benchmark in reinforcement learning research, it provides a simple yet effective environment to evaluate and compare the performance of various algorithms and it illustrates fundamental concepts, such as the trade-off between exploration and exploitation. Although it is a toy example, the Cliff Walking Problem mirrors real-world scenarios where agents must navigate risky environments, such as in robotics, autonomous vehicles, and game AI. Understanding how to manage risks and optimize rewards in these contexts is crucial for developing effective reinforcement learning solutions.

This essay aims to explore and compare different reinforcement learning approaches for solving the Cliff Walking Problem. We will start by discussing the Bellman equation and examine how it serves as a foundation for two more advanced algorithms: SARSA and Q-Learning. Through this exploration, we aim to identify the strengths and weaknesses of each algorithm and understand their applicability in various reinforcement learning scenarios.

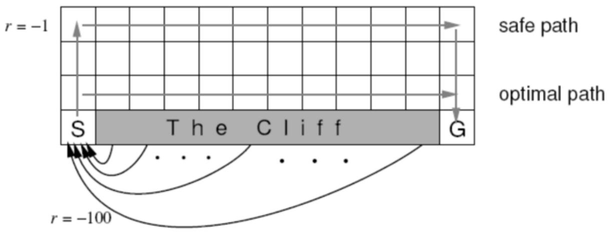
2. Description of the Cliff Walking Problem

In this section, we will provide a detailed overview of the Cliff Walking Problem, beginning with a definition of the environment in which the problem is set. We will focus on the critical components that govern the agent's interactions within the grid world, specifically the states, actions, policies, and rewards. The dynamic interplay among these elements shapes the agent's learning process and highlights the unique challenges of this problem, for which *Temporal Difference algorithms* prove to be particularly well-suited.

2.1 **Environment definition**

The Cliff Walking Problem is defined within a 4x12 grid world, where each cell represents a state *s* that the agent can occupy. A state is a specific configuration of the environment at a given time and the set of states can be defined as *S*, where $S = \{s0, s1, \ldots, s47\}$ for the possible 48 states in the grid.
An action is a decision made by the agent that affects its state and at each step, the agent can choose one of four actions from the action set *A={up ,right, down, left}*. Each action results in a transition to a new state, with certain actions potentially leading the agent closer to the goal or, conversely, falling into the cliff area. The objective is to navigate from the initial state *s*0 to the terminal state *s*47 while avoiding the cliff, which lies along the bottom edge of the grid.
The policy, denoted as $\pi: S \to A$, is the strategy the agent employs to determine which action to take in each state, which can be more deterministic (selecting the same action for a given state) or more stochastic (choosing actions based on probabilities) depending on the *ε-greedy* policy setting.
The Cliff Walking Problem features a specific reward structure that guides the agent's learning process towards the optimal policy π. The agent receives a reward of -1 for each step taken in any state that is not part of the cliff area. If the agent falls off, it will be returned to the starting position and incur a penalty of -100. Thus, we can summarize the environment and the rewards as:

$$R(s_i) = \begin{cases} -1 & \text{for all } s_i \text{ not on the cliff} \\ -100 & \text{for all } s_i \text{ on the cliff} \end{cases}$$



To sum up, the agent uses an initial policy to choose actions based on its current state and receives rewards that inform its learning process and improve its policy. The *optimal policy* is a strategy that maximizes the cumulative reward the agent receives while navigating from the initial state *s0* to the terminal state *s47* while avoiding the cliff.

2.2 **Temporal Difference Learning in the Cliff Walking Problem**

The Cliff Walking Problem serves as an ideal case study for *Temporal Difference* (TD) method in reinforcement learning. TD techniques allow an agent to predict future rewards based on past experiences without requiring a complete model of the environment or waiting for an entire episode to conclude. At the heart of the updating method is the concept of *temporal difference*, which is the gap between the agent's current value estimate and the newly observed value following each action. This difference drives the learning process, as the agent uses it to incrementally update its estimate towards the latest observation in real time. When the agent takes an action - often guided by a ε-greedy policy- it immediately adjusts its value estimate according to the observed reward and the next state. This real-time updating process is known as *bootstrapping* and promotes faster convergence toward optimal behavior.

The basic updating formula for TD learning can be expressed as:

$$V(s) \leftarrow V(s) + \alpha \left(r + \gamma V(s') - V(s)\right)$$

where *V(s)* is the current value estimate for state s, r is the reward received after transitioning to the next state, s', α is the learning rate, and γ is the discount factor that prioritizes immediate rewards over distant future rewards.

Temporal Difference learning is particularly well-suited for situations where the agent lacks complete prior knowledge of the environment and where immediate updates to value estimates can accelerate learning. In the Cliff Walking scenario, the agent starts without knowing which paths are safe and which lead to danger. Since it can only explore the environment incrementally, waiting for an entire episode to conclude before updating its value estimates would slow down learning. The incremental nature of TD learning allows the agent to adjust its estimates immediately, enhancing its ability to avoid the cliff and reach the goal efficiently.

Comparing TD learning with *Monte Carlo methods* and *Dynamic Programming* (DP) clarifies TD's unique strengths. Specifically, Monte Carlo methods require complete episodes to evaluate the total reward received and value estimates are updated only after an episode concludes. This dependence on full sequences leads to higher variance and slower convergence. Conversely, Dynamic Programming requires a model of the environment dynamics - specifically, the transition probabilities and expected rewards— and value estimates are computed through the *Bellman equation*. However, the assumption of prior knowledge of the environment's dynamics limits DP feasibility in real-world scenarios.
By integrating elements from both methods, TD learning offers a more flexible and efficient approach to learning from incomplete sequences. From Monte Carlo, TD inherits the ability to learn directly from experience by updating value estimates based on received rewards and extends it to incomplete sequences. It also adopts and enhances the Bellman equation from DP to compute expected rewards without requiring a comprehensive model and knowledge of the environment's dynamics, thus enabling incremental updates based on other learned estimates.

TD learning's adaptability and immediate feedback make it an ideal choice for complex, dynamic environments like Cliff Walking. In the following sections, we will explore the primary TD algorithms: SARSA and Q-Learning, including their advantages and limitations. Intuitively, one of the major challenges when applying TD is the careful tuning of parameters involved in the update rule. Proper tuning is essential, as it can significantly impact the agent's learning stability and convergence rate.

3. Bellman's Equation

3.1 Introduction to Bellman's equation.

Although SARSA and Q-learning do not use Bellman's Equation directly in its original form for updating value estimates, discussing Bellman's Equation remains essential because it is foundational to all reinforcement learning methods that utilize value functions.

Both SARSA and Q-learning rely upon the core principles introduced by Bellman's Equation: that a value function *V(s)* (for state-value) or *Q(s,a)* (for action-value) can represent the expected cumulative reward by recursively incorporating the value of future states. This recursive approach underlies not only how the agent evaluates states and actions but also how it incrementally learns from each experience to build towards optimal decision-making

3.2 **Mathematical formulation of Bellman's equation**.

As we will see in further chapters, SARSA and Q-learning both operate with the *Q-value function Q(s,a)*, which estimates the expected return for taking an action in a given state and then following a policy. Therefore, we'll describe the Bellman's Equation for *Q(s,a),* clarifying how current action-value estimates are derived from both immediate rewards and the anticipated future rewards of subsequent states

$$Q(s,a) = r + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

In this formulation:

1. *r* is the immediate reward received after transitioning to the next state.
2. γ is the discount factor, which balances the importance of immediate rewards against future rewards.
3. *P(s'/s,a)* is the transition probability, indicating the likelihood of moving to state *s'* after taking action *a* in state *s*
4. *max (a') Q(s',a')* represents the maximum expected future return for the next state *s'* considering all possible actions *a'.*

As the primary objective of any reinforcement learning approach is to reach the optimal policy, once the value function is established, the agent can improve its policy by choosing actions that maximize the expected return based on the current value estimates. This is often expressed as:

$$\pi'(s) = \arg\max_{a} Q(s,a)$$

The improved policy π′ selects actions that yield higher expected rewards, which can be derived from the value estimates computed using Bellman's Equation. The process of policy evaluation followed by policy improvement can be repeated iteratively and is known indeed as *policy iteration*, leading to increasingly better policies.

3.3 Limitations of Bellman's Equation in the Cliff Walking Environment

Despite its foundational role, Bellman's Equation has notable limitations in the context of the Cliff Walking environment. One significant challenge is the assumption of known *transition probabilities P(s'/s,a)*, which the equation relies on to compute expected values. In the Cliff Walking scenario, the agent lacks complete knowledge of the environment, making it difficult to accurately estimate these probabilities; thus, relying on them could lead to suboptimal decision-making.
Moreover, Bellman's Equation assumes a *static environment* where transition probabilities remain constant. However, in dynamic scenarios like Cliff Walking, the presence of cliffs introduces states where the expected rewards drastically change, creating discontinuities in the value function. When the value function experiences a significant drop upon entering a cliff state, the standard form of Bellman's Equation may not adequately reflect the severity of this change.
Therefore, while Bellman's Equation provides a critical theoretical framework, the model-free approaches of SARSA and Q-learning offer a more suitable strategy for facing the challenges presented by the Cliff Walking scenario. As we'll see in the next chapter, due to the lack of prior knowledge of the environment's dynamics, SARSA and Q-learning adapt the original formulation of the Bellman's equation by omitting transition probabilities.

4. SARSA and Q-Learning: two Temporal Difference algorithms

The following sections examine the strengths and limitations of the SARSA and Q-Learning algorithms in depth. However, before proceeding, it is essential to distinguish between on-policy and off-policy methods, this distinction serves as a foundation for understanding how these algorithms behave and can be implemented.

4.1 Concept of On-Policy (SARSA) vs. Off-Policy (Q-Learning) Methods

When discussing Temporal Difference (TD) learning and updating rules, it is crucial to distinguish between behavioral policies and updating policies (used to compute action-state values).
This distinction is relevant because the choice of updating policies and behavioral policies determines whether an algorithm is classified as *on-policy* or *off-policy*.
We have already mentioned that a policy denoted by $\boldsymbol{\pi}$ *i*s the probability of each action a ∈ A being taken for each state s ∈ S. The *behavioral policy* outlines how an agent acts in any given state, detailing the probabilities of choosing each possible action. In contrast, the *updating policy* or *learning policy* defines how the agent predicts its future actions when assessing the value of a state-action pair.

On-Policy Learning **:** An *on-policy* method like SARSA focuses on learning the action-value estimates based on the agent's current policy. This means that the agent learns by following and improving upon the same policy it uses to make decisions in the environment. This approach is beneficial in environments where the agent should avoid risky actions, it produces more conservative policies that align with actual experiences and that are influenced by the actions taken.
*Off-Policy Learning***:** In contrast, *off-policy* methods like Q-Learning learn the value of the optimal policy independently of the agent's current policy. This approach is more optimistic as it assumes that an optimal action is made in every state and therefore it updates its state-action estimates using the maximum expected value for the next state.

To summarize, in on-policy methods, the policy for action selection (behavioral policy) and the policy for learning (updating policy) are one and the same. Conversely, in off-policy methods, they differ, with the learning policy often being a greedy policy that seeks to maximize future rewards.

4.2 SARSA and Q-Learning updating rules

Both SARSA and Q-Learning are *control algorithms* designed to learn the optimal policy for controlling the agent's behavior, with the optimal goal of maximizing expected returns. While both algorithms utilize value functions to assess their decision-making processes, they employ different updating rules.

SARSA Action-Value Function and updating rule
The name SARSA is derived from the sequence of components it considers: state, action, reward, next state, and next action. The action-value function for SARSA is defined as follows:

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

Where: α is the learning rate, *r* is the immediate reward received after taking action a in state *s, γ* is the discount factor, *s′* is the next state and *a′* is the action taken in state 's′ according to the current policy π.

Q-Learning Action-Value Function and updating rule
The action-value function for Q-Learning is defined as follows:

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

In this equation, the term *max_a′ Q(s′,a′)* represents the value of the best possible action in the next state s′, irrespective of the current policy.

The primary distinction between SARSA and Q-Learning lies in how they update their action-value functions, particularly in the temporal difference component, which is linked to the on-policy and off-policy nature of the two algorithms.

The temporal difference for SARSA is given by: $\mathrm{TD}_{\mathrm{SARSA}} = r + \gamma Q(s', a') - Q(s, a)$

The same difference in Q-Learning is given by: $\mathrm{TD}_{\text{Q-Learning}} = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$

The way each algorithm updates its action-value estimates influences not only the *speed of learning* but also the *stability* of the policies they develop. In the following section, we will explore how the differing updating mechanisms affect the behavior and effectiveness of the two algorithms.

4.3 Strengths and weaknesses of Sarsa and Q-Learning

SARSA and Q-Learning each offer distinct advantages and challenges that influence their performance in various environments
As an on-policy algorithm, SARSA is particularly sensitive to the experiences it gathers during training. It updates its action-value estimates based on the agent's current policy, promoting a cautious and adaptive learning process. This safety-focused approach provides stability and makes SARSA suitable for scenarios where avoiding high-risk actions is essential. Although SARSA may converge more slowly toward the optimal policy—since its decision-making improvements occur at the end of each episode—it typically results in stable average returns.
In contrast, Q-Learning operates as an off-policy method, which promotes faster convergence towards the optimal policy. Indeed, optimizing its action-value estimates independently of the current policy implies that the current policy is improved at each step and not at the end of the episode.
While this flexible approach results in faster convergence it can introduce higher risks and unstable average returns, particularly in the initial stages of training. This instability occurs because the agent takes risky actions, leading it to "fall off the cliff".

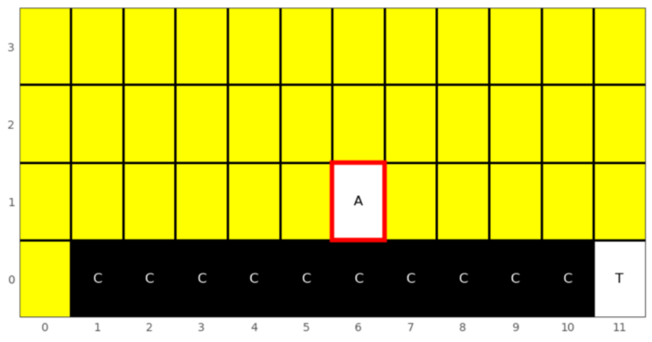
A question arises regarding *whether the optimal policies found by SARSA and Q-Learning coincide*. There is indeed a scenario in which both algorithms converge to the same solution. The subsequent chapters will delve into the implementation of these algorithms, further illustrating their fundamental differences, answering our questions and confirming our expectations about their overall performance and behavior.

5. Implementation of SARSA and Q-Learning:

5.1 Setting up the Cliff Walking Environment

Setting up the Cliff Walking environment requires the use of libraries designed to create and manage reinforcement learning environments.
To set up the Cliff Walking environment, we use the '`gym`' library. Specifically, the `CliffWalkingEnv` class initializes a 4x12 grid with key default parameters: a starting position, cliff locations, and defined actions.

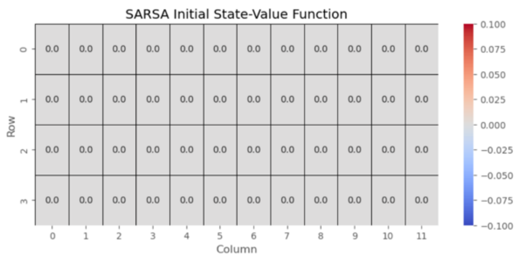
Actions are encoded as follows: `action = [0: "up", 1: "right", 2: "down", 3: "left"].`
The agent begins in the bottom-left corner and must reach the goal in the bottom-right corner (labeled "T") while avoiding a "cliff" along the bottom row. If the agent steps onto a cliff cell, it receives a large negative reward and is reset to the starting position. Before each episode, the environment is reset to ensure it begins from this initial state, allowing consistent and reliable training for the agent.



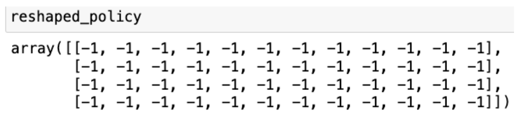
5.2 Brief explanation of the SARSA implementation.

To implement the SARSA algorithm, we begin by initializing the action-value function Q with zero values for all state-action pairs, ensuring a uniform starting point for learning. [Fig.1]
Additionally, we create an object called `stats`, which tracks episode rewards and lengths, providing insight into the agent's performance over time
We then define the policy as epsilon-greedy through the function `make_epsilon_greedy_policy` and we initialize the policy to "-1 : no action" [Fig.2]
The policy function ensures that for each action choice, the agent will either explore a random action with a probability of epsilon, or exploit by choosing the best-known action. This balance helps the agent avoid suboptimal paths and progressively learn the most rewarding actions

[Fig.1]



[Fig.2]



```
reshaped_policy
array([[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
       [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
       [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
       [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]])
```

The main SARSA implementation is structured in two nested loops:

- In the outer loop, for each episode, we initialize the state and select an initial action according to the epsilon-greedy policy.
- In the inner loop, for each step of the episode, the agent:
  1. chooses A' from S' using policy derived from Q
  2. chooses the next action A′ from S′ based on the same epsilon-greedy policy, using the Q-values for guidance.

The Q-values will be updated using the SARSA update rule which is :
$Q(S,A) := Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

In the code, this is represented by:

```
td_target = reward + discount_factor * Q_decay[next_state][next_action]
td_error = td_target - Q_decay[state][action]
Q_decay[state][action] += alpha * td_error
```

The SARSA update rule adjusts the Q-value for the current state-action pair by moving it closer to the "target" value, which is the reward plus the estimated future reward from S′ under A′. By doing so, SARSA incorporates the immediate reward and the future expectation in an on-policy manner, as it considers the action actually chosen from S′.

The episode statistics are updated with each reward and length, allowing us to track how well the agent performs over time. The loop will break and the episode will end if the agent reaches a terminal state. At each step, the current state and action variables are updated to the next state and action, in other words : S <- S’ and A<- A’.

An important addition to our SARSA implementation is the "`epsilon-decay`" parameter – together with a specified minimum epsilon value (`min_epsilon`) - which allows an early exploration at the beginning and a gradual shift to exploitation as training progresses. An initial higher epsilon ensures the agent does not get stuck in suboptimal policies because it explores a wider range of actions.
If the agent exploits too soon (without enough exploration), it might settle on a suboptimal policy that doesn't maximize long-term rewards. Conversely, by gradually decaying epsilon, the agent increasingly focuses on actions that have shown high rewards, leading to faster convergence to an optimal or near-optimal policy. All in all, the gradual decay allows to achieve a good balance between exploration and exploitation.

5.3 Brief explanation of the Q-Learning implementation.

As for Q-learning algorithm, the initialization of the action-value function Q and the policy is the same as what has been previously described for SARSA.

The implementation with the two nested loop is very similar except for how the algorithm behaves when it updates the Q-values. In the Q-Learning algorithm, the Q-values will be updated using an off-policy formula which is:

$$Q(S,A) := Q(S,A) + \alpha[R + \gamma \text{max\_a } Q(S',A') - Q(S,A)]$$

In the code, this is represented by:
```
td_target = reward + discount_factor * np.max(Q_decay[next_state])
td_error = td_target - Q_decay[state][action]
Q_decay[state][action] += alpha * td_e
```

This update modifies the Q-value for the current state-action pair by adjusting it closer to the "target" value, which is the immediate reward plus the estimated maximum future reward from S′. By doing so, Q-Learning incorporates both the immediate reward and the expected future rewards in an off-policy manner, focusing on the best possible action from S′ regardless of the action actually taken.

The remaining aspects of the Q-Learning implementation, including the updates to episode statistics and the handling of the epsilon decay policy, are equivalent to those already described for the SARSA algorithm.

6. Results and Performance Comparison

6.1 Comparison of the performance of SARSA and Q-Learning

The results demonstrate the distinct characteristics of SARSA and Q-Learning, as observed in their *state-action value matrices*, *policy representations*, *episode lengths over time*, and *cumulative rewards per episode.* These visual distinctions emphasize SARSA's inclination toward safer policies, while Q-Learning’s policy reflects a commitment to efficiency, even at the potential cost of safety

The *state-action value matrices* (Fig 3 and Fig 5) illustrate the final optimal state-action values, which indicate the expected returns for the agent following the optimal policy from any given state. Each algorithm demonstrates distinct approaches to risk management.

SARSA, as an on-policy algorithm, updates its Q-values based on the actions actually taken under its current policy. This approach incorporates risk considerations, leading to maximum Q-values that are generally lower than those of Q-Learning. SARSA's cautious strategy promotes safer paths, often steering the agent away from dangerous areas like cliff edges. Consequently, it assigns higher values to states which are far from the cliff and very low state-action values for states near the cliff, favoring less risky routes even if they are longer.
In contrast, Q-Learning, which is off-policy, updates its Q-values using the highest value action in the next state, regardless of the current policy. This method is more aggressive and prioritizes reward maximization over caution. As a result, Q-Learning typically converges to higher state-action values overall, assigning greater values to states closer to the cliff and lower values to states far from the cliff.

The *policy representations* (Fig 4. and Fig 6.) further emphasize these tendencies (the policy is represented using arrows for clearer understanding).
Consistently with the Q-values matrices, SARSA's final policy avoids states near the cliff edge, choosing actions that navigate away from risk, such as consistently moving "up" in dangerous areas, while Q-Learning favors direct routes toward higher rewards, even if they approach perilous cliff edges. All in all, SARSA's optimal policy is sub-optimal because a faster and safe path exists.

Fig 3.

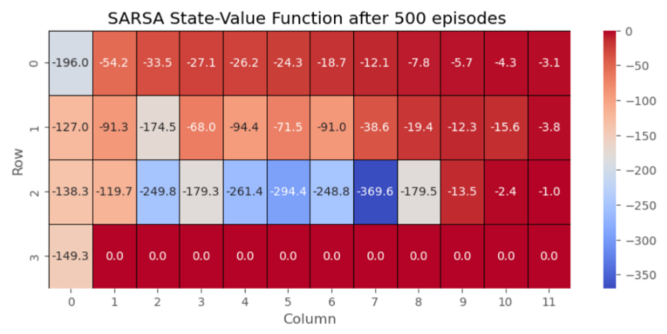


Fig 4.

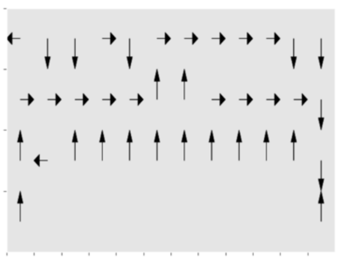


Fig 5.

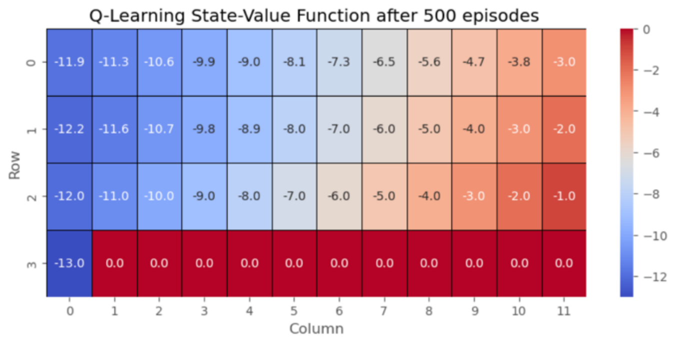
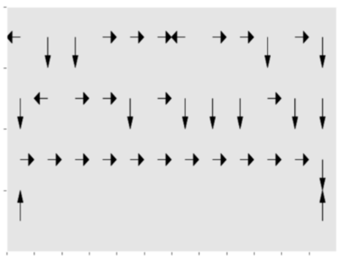


Fig 6



Here are two graphs that provide additional insights into the performance of SARSA and Q-Learning in the cliff-walking environment. The first graph illustrates the episode lengths over time, while the second graph showcases the total rewards accumulated per episode for each algorithm.

SARSA yields a reliable but potentially slower path towards convergence. Consequently, episode lengths are significantly higher during the first 0-100 episodes and start to decrease steadily thereafter. In contrast, Q-Learning achieves shorter episodes faster, reflecting its off-policy, value-maximizing strategy that allows for quicker adaptations to the environment.

In terms of reward, Q-Learning typically achieves higher total rewards per episode. However, the variability in rewards is also higher because the agents falls many times off the cliff. SARSA's rewards tend to be lower than Q-Learning, but they are more consistent and stable, reflecting its safer exploration strategy.

Notably, the average rewards stabilize after a certain number of episodes: SARSA stabilizes around -17, while Q-Learning stabilizes around -13. This indicates that Q-Learning reaches the terminal state more quickly, as it traverses fewer states overall.

SARSA:

Fig. 7

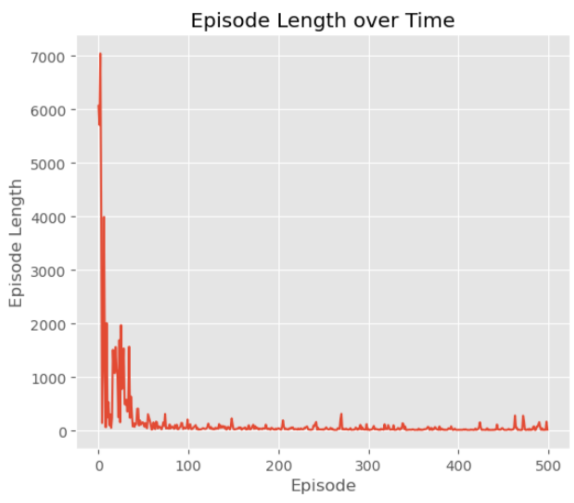
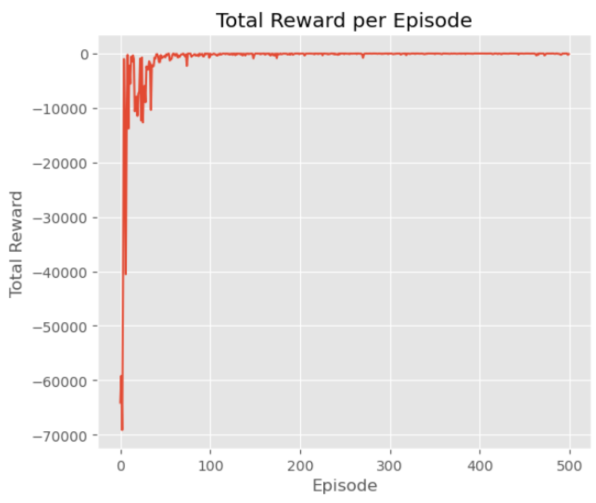


Fig 8.



Q-LEARNING:

Fig.9

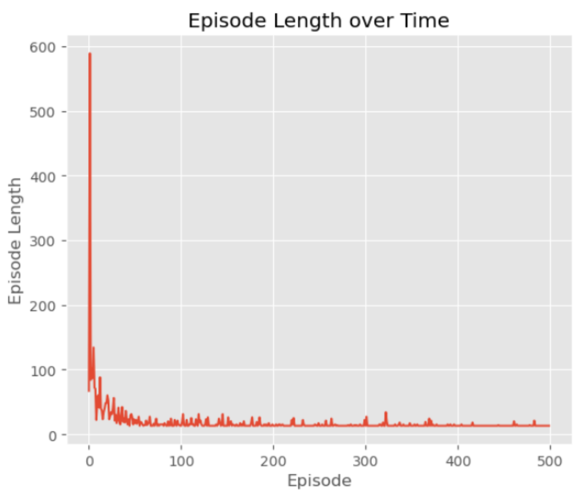
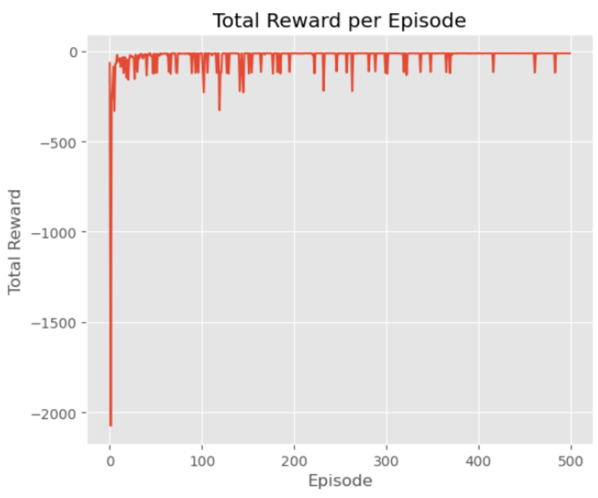


Fig.10



In summary, SARSA proves to be more stable and prudent, suitable for environments where avoiding high-risk actions is essential. However, this caution can lead to slightly sub-optimal performance, as SARSA may bypass faster paths that carry more risk. Q-Learning, on the other hand, converges more quickly to the optimal policy due to its off-policy nature, which maximizes value regardless of short-term risks. This makes Q-Learning particularly effective in environments where a "higher risk-higher return" strategy is appropriate. Ultimately, the choice between SARSA and Q-Learning will depend on the desired balance between optimal performance and safety in the specific environment the agent operates within.

6.2 Special case: 100% greedy policy

When a 100% greedy policy is employed, both SARSA and Q-Learning converge to the same optimal solution. In this scenario, the agent consistently selects actions that maximize the current estimate of the Q-values, regardless of the exploratory nature of the algorithms, in fact, exploration is minimized.
For SARSA, this means that the agent will only take the highest-value action as dictated by its current policy, leading to a direct path toward the optimal policy without exploring less optimal alternatives. In this case, SARSA behaves similarly to Q-Learning, as it updates its Q-values based solely on the actions that yield the highest rewards.

This special deterministic setting reinforces the idea that the essential difference between SARSA and Q-Learning lies in their exploration strategies rather than in their convergence behavior. Therefore, the decision to use SARSA or Q-Learning is more about the agent's exploration approach and how it interacts with the environment rather than the algorithms' capabilities to find optimal solutions.

As shown by the pictures, when the approach is 100% greedy, we find similarities not only in the optimal policy pattern, but also in the final optimal state-action matrices of the two algorithms

Fig. 11

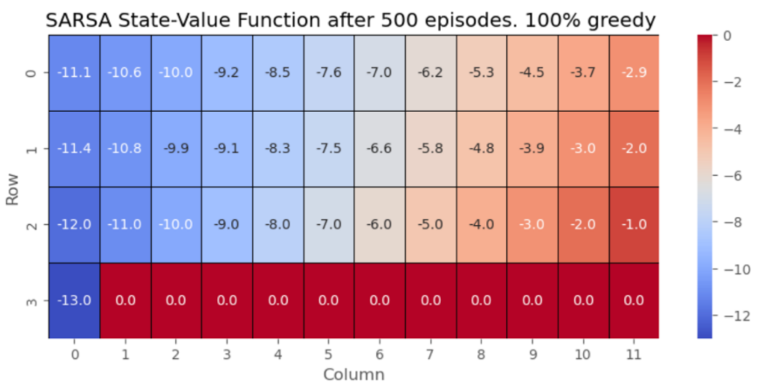


Fig 12.

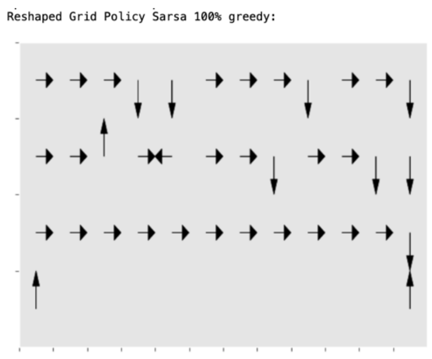


Fig. 13

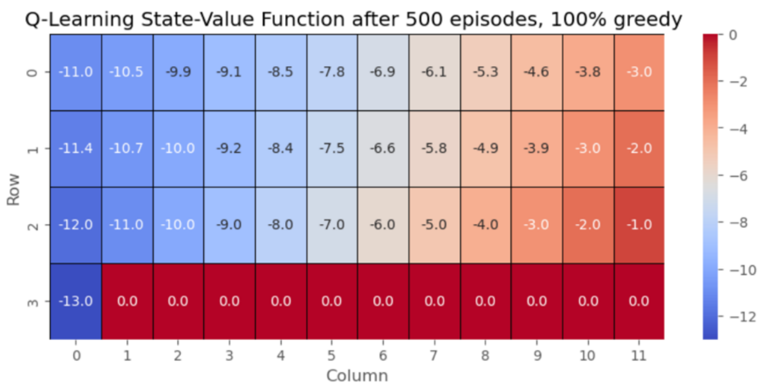
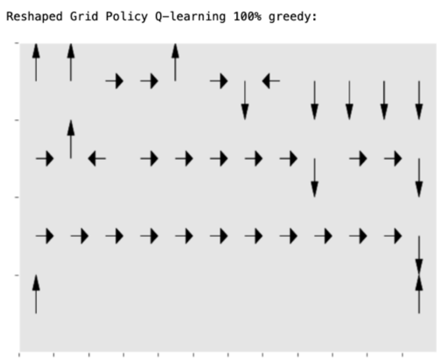


Fig 14.



6.3 Limitations of TD and future development of the Cliff Walking Problem

Both SARSA and Q-Learning, as well as Temporal Difference (TD) learning methods in general, face several limitations when applied to the Cliff Walking Problem.
A primary challenge is the *difficulty in effectively learning from the steep penalties* associated with falling off the cliff. In other words, the cliff penalty may not be fully reflected in its value estimates. This can lead to either excessive caution, where the agent avoids beneficial exploration, or insufficient caution, resulting in risky behaviors.

Another challenge arises from the *frequency of penalty encounters*. If the agent does not experience these negative rewards often enough, it may not recognize the critical importance of avoiding the cliff. All these drawbacks can lead to suboptimal value estimates and extended learning periods.
To mitigate these issues, fine-tuning of TD learning parameters is crucial. Techniques such as *reward shaping* can provide more immediate feedback, enhancing the learning process.

Future research could focus on *modifying the grid size*, *adjusting the reward structure*, *or introducing obstacles* to better understand the adaptability of SARSA and Q-Learning in different scenarios.
By addressing these limitations, researchers can enhance the efficiency of TD learning methods in complex environments like the Cliff Walking Problem.

7. References

[1] Sutton, R. S., & **Barto**, A. G. (2018). **Reinforcement learning**: An introduction (2nd ed.)
[2] https://github.com/vBarbaros/medium-py/blob/main/cliffwalking-env/cliffwalking-with-sarsa.ipynb?source=post_page-----95ca57735a8f--------------------------------
[3] https://towardsdatascience.com/reinforcement-learning-cliff-walking-implementation-e40ce98418d4