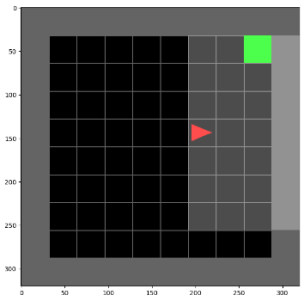
Report on Reinforcement Learning Strategies in MiniGrid Environments

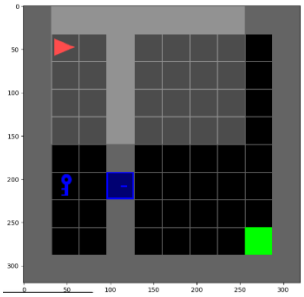
Introduction

In this comprehensive exploration, we delve into the nuances of applying classical reinforcement learning (RL) techniques within the MiniGrid environment. Targeting two distinct challenges—`RandomEmptyEnv_10` and `RandomKeyMEnv_10`—our objective transcends mere problem-solving to unearth deeper insights into the dynamics of various RL strategies.

Environment Overview

MiniGrid offers a broad spectrum of tasks, from simple navigation to complex object interactions, providing an ideal testing ground for RL algorithms. Our focus lies on two specific environments:





- **RandomEmptyEnv_10:** Here, the agent's task is to navigate a maze to reach a designated goal. This environment tests the agent's learning and adaptation capabilities to efficiently find the goal.

- **Q-Table Size Consideration:** For `RandomEmptyEnv_10`, the environment is simpler, with the goal being in one of three possible locations. Given that the agent has 63 potential positions (excluding the goal positions) and can face in one of four directions at each position, the total number of states is calculated as 3 X 63 X 4 = 756. With three possible actions (rotate left, rotate right, and go forward), so the **Q-table size = 756 X 3**, the complexity and computational demands are relatively manageable, providing a clear framework for RL exploration.

- **RandomKeyMEnv_10:** This environment presents a higher level of complexity, requiring the agent to find a key, unlock a door, and then reach the goal. This sequence adds layers to the learning task.

- **Q-Table Size Consideration:** The Q-table size dramatically increases due to the sequential nature of tasks and the variability in the environment's state. Initially, the goal's position is fixed, and the wall is always on column 3. The door can be in one of eight positions, and we distinguish between scenarios before and after the key is acquired. Prior to obtaining the key, with 15 possible locations for the key (left to the wall) and 15 possible agent positions (excluding the key's location), multiplied by four directions, the state-space begins to expand. After the key is acquired, the number of agent positions depends on whether the door is open or closed, leading to a total of 8 X ((15 X15 X 4) + (56 X 4 + 55 X 4) = 10,752 states, factoring in the five possible actions (rotate left, rotate right, pick up, toggle, go forward), so the **Q-table size = 10,752 X 5**. This extensive state-space highlights the heightened complexity and learning challenge presented by `RandomKeyMEnv_10`.

These environments, with their discrete state and action spaces and complete observability, align perfectly with the Markov Decision Process (MDP) framework essential for RL.

Algorithmic Approach and Hyperparameter Tuning

Our methodology involved an exhaustive training regimen for Monte Carlo (MC), SARSA, and Q-Learning algorithms, with a keen eye on hyperparameter optimization. Key configurations included:

- **Gamma (Discount Factor):** Opted for ` [0.9, 0.99] `, affecting the agent's foresight in valuing future rewards.

- **Alpha (Learning Rate):** ` [0.1, 0.2] ` for SARSA and Q-Learning, guiding the magnitude of Q-value updates.

- **Initial Epsilon:** ` [0.5, 1] `, setting the exploration rate outset, where `init_epsilon` of `1` markedly outperformed `0.5`, fostering more initial exploration and thus leading to quicker, smoother convergence.

- An `**epsilon_decay**` rate of `0.999` was uniformly applied to facilitate a dynamic equilibrium between exploration and exploitation as learning advanced.

State Representation and Q-Table Key Hashing

An integral aspect of our algorithmic implementation involves the encoding and utilization of the environment state for reinforcement learning. The MiniGrid environment provides an observation that includes an array of numbers, where every three numbers define a square in the environment. This observation array encompasses the entire state space, including the agent's direction, effectively making the observation synonymous with the state in our context.

To facilitate efficient state management and retrieval within our Q-learning framework, we transformed this array into a hashable data structure. Specifically, we converted the array into a tuple, which serves as a key in our Q-table. This conversion is crucial for two main reasons:

1. **Hashability:** Tuples, unlike arrays, are immutable and hashable in Python. This characteristic allows us to use them as keys in a dictionary (our Q-table), enabling fast lookup, insertion, and deletion operations. This efficiency is pivotal in reinforcement learning, where numerous state-action pairs must be accessed and updated iteratively.

2. **State Uniqueness:** By encoding the entire observation array into a tuple, we ensure that each unique state of the environment is represented distinctly in the Q-table. This distinction is vital for the learning algorithm to correctly associate actions and rewards with the precise state configurations they originate from.

This representation strategy aligns with our broader algorithmic approach, ensuring that the complexity of the MiniGrid environment's state space is adequately captured and efficiently managed. It allows our reinforcement learning models to effectively discern and learn from the diverse state configurations encountered during training.

Strategic Reward Shaping in RandomKeyMEnv_10

The intricate design of `RandomKeyMEnv_10` demanded a nuanced application of reward shaping strategies to effectively navigate its complexities. Our approach was twofold, carefully balancing incentives for desirable behaviors with deterrents for counterproductive actions. Specifically, we implemented a mechanism that not only rewards the agent for door-opening actions but also introduces a penalty for closing doors. This nuanced reward structure was essential for several reasons.

Firstly, by assigning a minor reward (0.001) for successfully opening a door, we directly encouraged the agent to learn and repeat the sequence of actions leading to door opening, a crucial step towards achieving the primary objective. However, to counteract potential exploitation of this reward—where an agent might repetitively open and close a door to accumulate rewards—we concurrently instituted a small penalty (also -0.001) for door-closing behaviors. This penalty dissuaded the agent from engaging in an endless loop of opening and closing the door, ensuring that the agent's actions are aligned with the overarching goal of the task.

Moreover, the magnitude of these rewards and penalties was deliberately chosen to be significantly lower than the final reward for reaching the goal (1.0), ensuring that while these shaping rewards guide the learning process, they do not overshadow the primary objective of reaching the goal. This balance is crucial for fostering a learning environment where the agent is motivated to explore and understand the key dynamics of the task—namely, the importance of the door as an obstacle and a gateway to the goal—without becoming fixated on the intermediate rewards.

This strategic reward shaping, in concert with our carefully selected gamma values, underscored our commitment to guiding the agent's learning trajectory towards efficient goal attainment. By rewarding door-opening and penalizing door-closing, we subtly steered the agent through the environment's challenges, ensuring that its learning and decision-making processes were tightly aligned with the ultimate task objectives. This approach not only enhanced the agent's performance in navigating the `RandomKeyMEnv_10` environment but also provided valuable insights into the effective design of reward structures in complex reinforcement learning scenarios.

Insights and Observations

Key insights from our experiments include:

- **Hyperparameter Impact:** Gamma, alpha, and initial epsilon selection profoundly influenced the learning process. Higher gamma values were notably effective in `RandomKeyMEnv_10`.

- Algorithm Performance: Q-Learning showcased superior performance with smoother learning curves compared to the more erratic SARSA and the swift but unstable Monte Carlo.

- Reward Shaping Efficacy: The reward shaping strategy in `RandomKeyMEnv_10` successfully guided the agent through the task's sequential challenges.

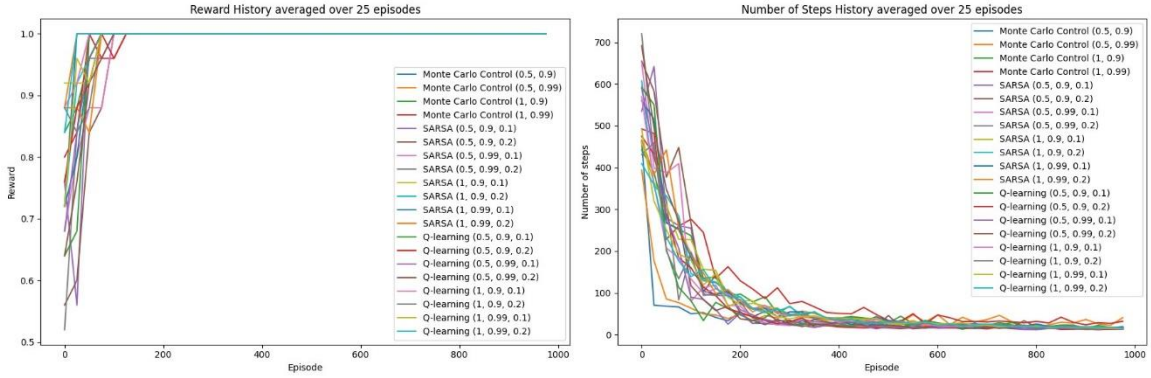
Results and Analysis with Hyperparameter Insights

Our analysis employs a structured array of plots for each environment, highlighting algorithmic performance against varying hyperparameters and providing a granular view into each strategy's effectiveness. Key insights are drawn from these visual representations, with particular attention to the convergence patterns and the impact of hyperparameter settings noted as (init_epsilon, gamma, alpha) on learning dynamics.

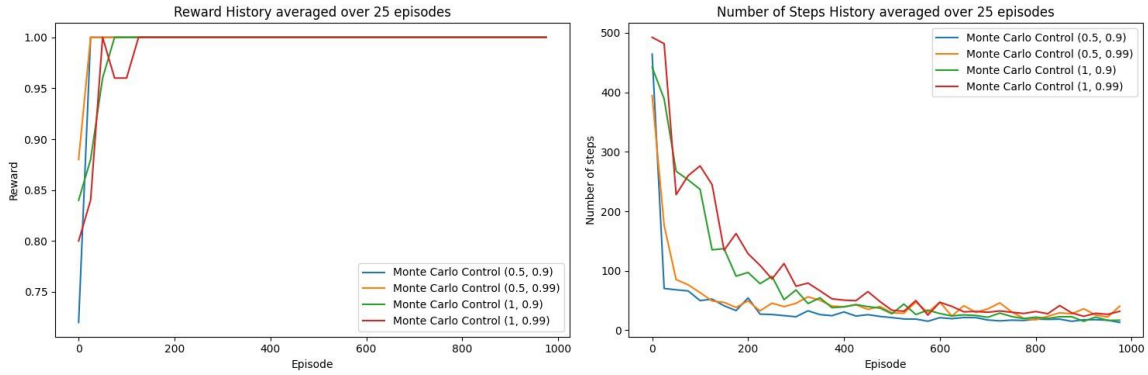
Analysis of RandomEmptyEnv_10

This segment distills the learning behaviors of each algorithm within the simpler maze navigation task, underscored by hyperparameter variations.

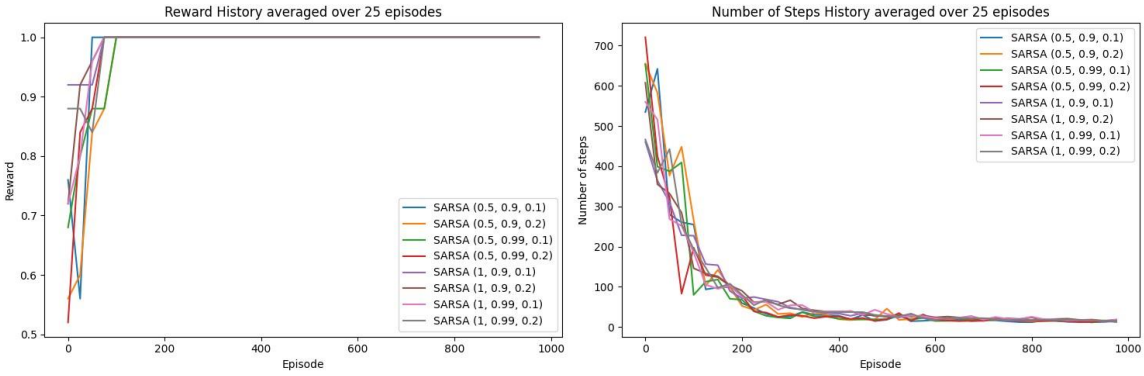
1. **Comprehensive Hyperparameter Impact:** A unified plot showcases the trajectories of Monte Carlo, SARSA, and Q-Learning algorithms across a range of hyperparameter settings. Remarkably, all algorithms demonstrate convergence around episode 200-400 (within a 1000 max steps framework).



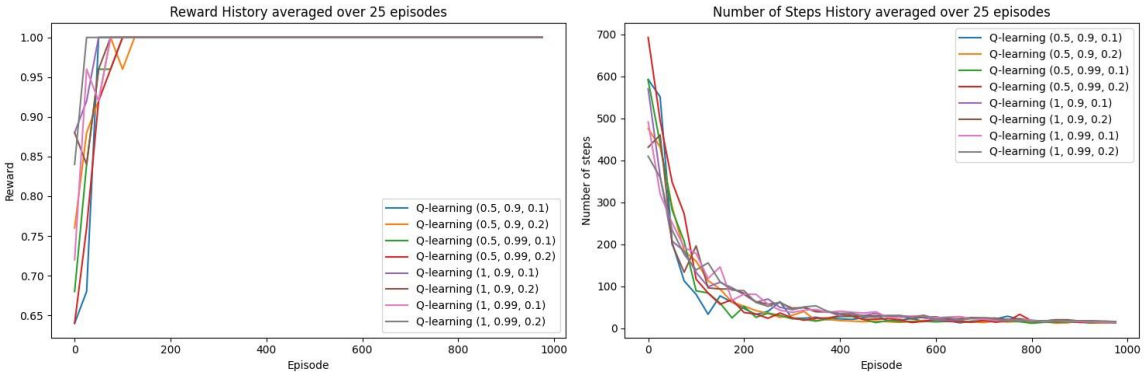
2. **Monte Carlo Hyperparameter Dynamics:** This analysis focuses on the Monte Carlo algorithm, highlighting how specific hyperparameters influence its rapid learning. The graph demonstrates the need for balanced settings to smooth its erratic progression, with an initial epsilon value of 1 notably leading to faster and smoother convergence.



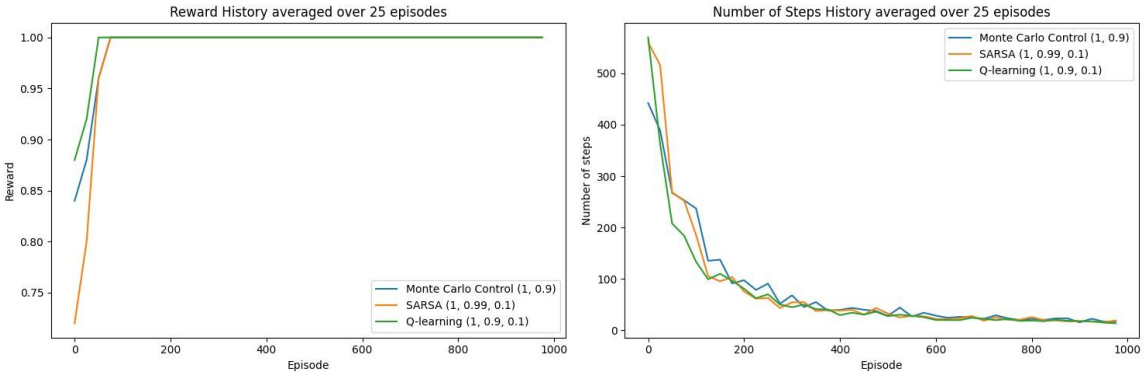
3. **SARSA Hyperparameter Variability:** SARSA's performance, charted across different settings, showcases its fluctuating learning curve. The plot vividly illustrates SARSA's sensitivity to hyperparameters, with its spiky trajectory signaling the challenge of finding optimal settings for stability.



4. **Q-Learning's Adaptability:** Focused on Q-Learning, this plot validates its consistent and stable improvement, emphasizing the algorithm's resilience. It becomes clear that Q-Learning benefits significantly from an optimized combination of `init_epsilon`, `gamma`, and `alpha`, underscoring its superior adaptability.



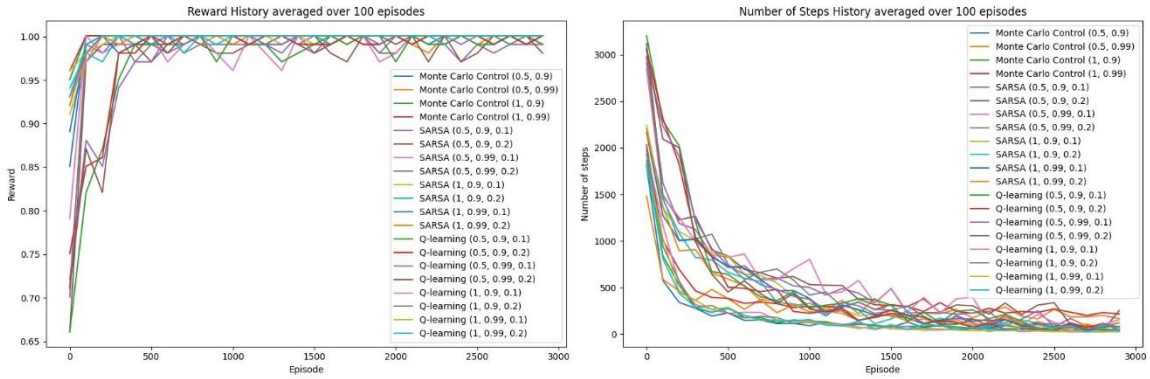
5. **Optimal Model Face-off:** Directly comparing the highest-performing models from each algorithm, this plot confirms Q-Learning's dominance in stability and learning efficiency within the `RandomEmptyEnv_10`, marking it as the standout approach for this environment.



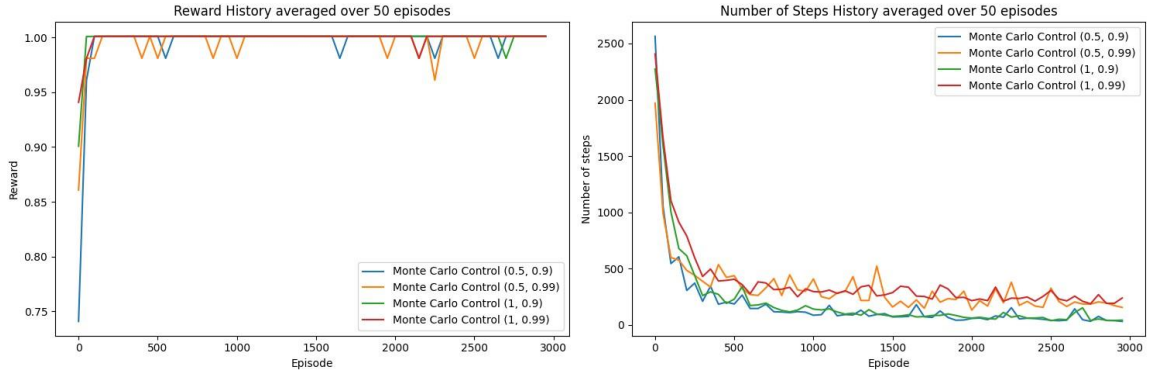
Analysis of RandomKeyMEnv_10

In the more complex `RandomKeyMEnv_10` environment, the plots delve into the intricacies of navigating tasks involving key retrieval and door unlocking.

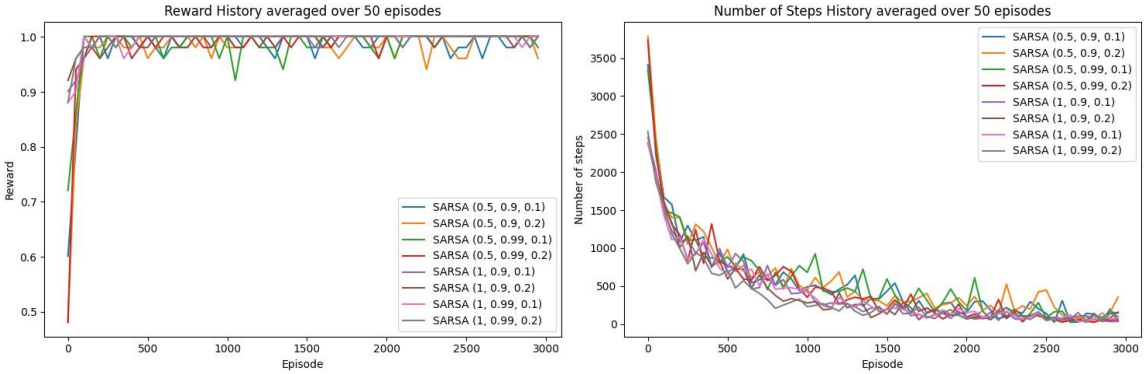
1. **All-Encompassing Hyperparameter Analysis:** This plot juxtaposes all three algorithms' performance under a myriad of hyperparameter combinations. Q-Learning's smoothness starkly contrasts with SARSA's volatility, providing a visual testament to the strategic advantage of careful hyperparameter selection in managing the environment's complexities, with the plot elucidating the pivotal role of `init_epsilon` in achieving swift and effective learning.



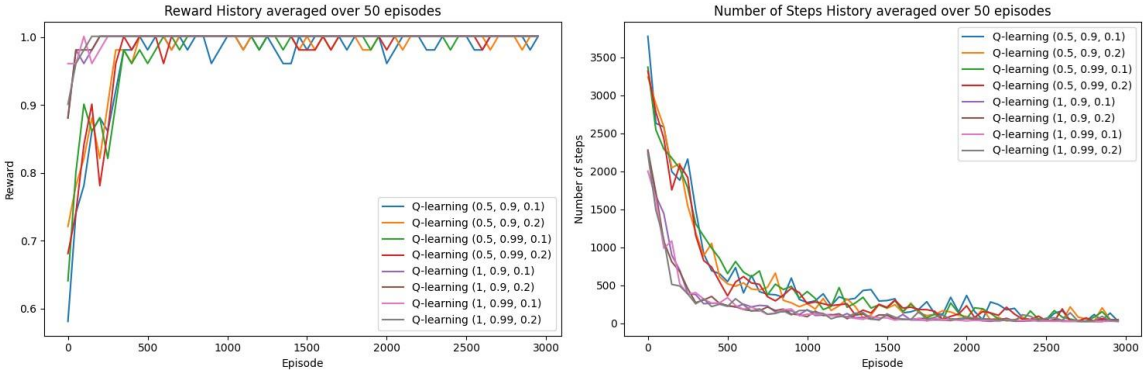
2. **Monte Carlo's Quick Adaptation:** Highlighting Monte Carlo's quick adaptation to RandomKeyMEnv_10, the plot emphasizes the need for balanced hyperparameters to stabilize reward fluctuations. Convergence occurs around 500 episodes (within a 5000-step maximum parameter). Notably, a gamma value of 0.9 performs smoother and better than 0.99, as evident in the graph.



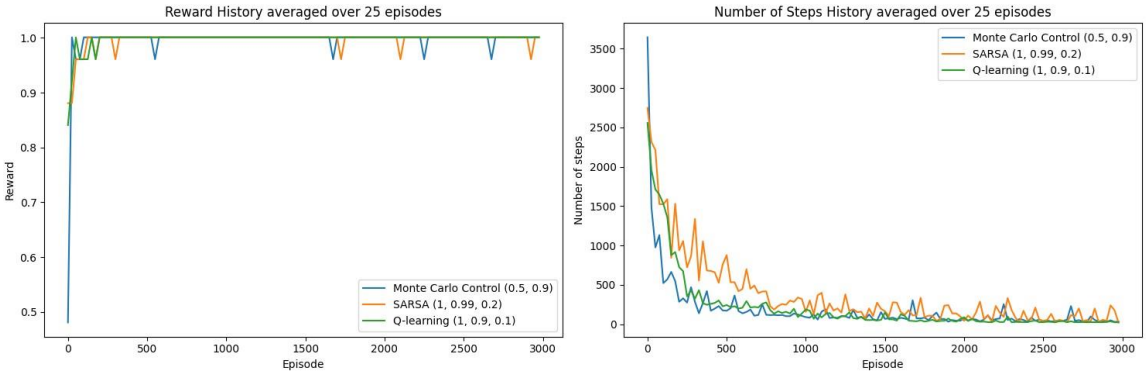
3. **SARSA's Learning Pattern:** Through a targeted examination of SARSA, its fluctuating and spiky learning curve is illustrated, showcasing a slower convergence typically spanning around 1000 episodes (with a maximum step setting of 5000). This underscores the difficulty of hyperparameter optimization for achieving stability. Notably, when the init_epsilon is set to 0.5, the curve exhibits pronounced spikes.



4. **Q-Learning's Consistent Progression:** Q-Learning's depiction within RandomKeyMEnv_10 underscores its smooth and resilient learning curve, marked by noticeable convergence around 500 episodes. This reaffirms its efficacy and adaptability in navigating complex scenarios. Notably, when the init_epsilon is set to 0.5, the curve exhibits slight spikes, whereas setting it to 1 results in a smoother and improved performance.



5. **Top Model Convergence Comparison:** In this conclusive comparison plot, the top-performing models of Monte Carlo, SARSA, and Q-Learning are juxtaposed, showcasing Q-Learning's unparalleled smoothness and faster convergence as the optimal strategy for `RandomKeyMEnv_10`. Notably, SARSA exhibits the least stability, characterized by frequent spikes in its performance curve.



Through these analyses, we offer a comprehensive view into how each algorithm fares across distinct MiniGrid challenges, illuminated by hyperparameter impacts. The visual data not only showcases learning efficiencies and patterns but also guides towards optimal algorithm-hyperparameter configurations for tackling specific environment challenges.

Optimal Training Parameters and Inference Performance

Our rigorous analysis and extensive training sessions have led to the identification of optimal hyperparameters for each algorithm within the `RandomEmptyEnv_10` and `RandomKeyMEnv_10` environments. Additionally, we pinpoint the best-performing algorithms in inference, showcasing their efficacy in navigating the complexities of each environment.

RandomEmptyEnv_10

```
Monte Carlo Control
Best hyperparameters: (1, 0.9) = (epsilon, gamma)
Best average reward: 0.99
Best average number of steps: 20.52
Best average done: 0.99

SARSA
Best hyperparameters: (1, 0.99, 0.1) = (epsilon, gamma, alpha)
Best average reward: 1.0
Best average number of steps: 9.18
Best average done: 1.0

Q-learning
Best hyperparameters: (1, 0.9, 0.1) = (epsilon, gamma, alpha)
Best average reward: 1.0
Best average number of steps: 9.37
Best average done: 1.0
```

Inference Performance Highlight:

```
Best Policy for Empty Environment
Best hyperparameters: ('SARSA', (1, 0.99, 0.1)) = (algorithm, (epsilon, gamma, alpha))
Best average reward: 1.0
Best average number of steps: 9.594
Best average done: 1.0
```

RandomKeyMEnv_10

```
Monte Carlo Control
Best hyperparameters: (0.5, 0.9) = (epsilon, gamma)
Best average reward: 0.98
Best average number of steps: 46.3
Best average done: 0.98

SARSA
Best hyperparameters: (1, 0.99, 0.2) = (epsilon, gamma, alpha)
Best average reward: 0.99
Best average number of steps: 32.3
Best average done: 0.99

Q-learning
Best hyperparameters: (1, 0.9, 0.1) = (epsilon, gamma, alpha)
Best average reward: 1.0
Best average number of steps: 22.85
Best average done: 1.0
```

Inference Performance Highlight:

```
Best Policy for Key Environment
Best hyperparameters: ('Q-learning', (1, 0.9, 0.1)) = (algorithm, (epsilon, gamma, alpha))
Best average reward: 1.0
Best average number of steps: 23.448
Best average done: 1.0
```

Conclusion on Hyperparameter Optimization and Performance insights

Our thorough investigation into hyperparameter optimization highlights the critical role that meticulous parameter tuning plays in amplifying the effectiveness of reinforcement learning algorithms across distinct MiniGrid environments. The strategic adjustment of hyperparameters, particularly the initial epsilon value, has proven to significantly influence the learning outcomes and efficiency of each algorithmic approach.

Through extensive experimentation, it became evident that an initial epsilon value of 1 generally yielded superior performance across both environments, facilitating smoother learning curves and more robust exploration strategies from the outset. This higher initial exploration rate allowed algorithms to understand and navigate the environments, leading to quicker convergence and more stable learning trajectories more comprehensively.

In the simpler **RandomEmptyEnv_10**, SARSA demonstrated unparalleled precision and efficiency, leveraging its optimal hyperparameter configuration to excel in task completion. The algorithm's success in this environment can be attributed to its ability to finely balance exploration with exploitation, guided by the strategic selection of hyperparameters, including an initial epsilon of 1.

Conversely, in the more intricate **RandomKeyMEnv_10** setting, Q-Learning emerged as the algorithm of choice, showcasing exceptional adaptability and robustness. Its standout performance is largely due to its effective utilization of a higher initial epsilon, enabling the algorithm to navigate the added complexities of key retrieval and door unlocking with remarkable efficiency. Q-Learning's smooth and consistent progress, underpinned by optimal hyperparameter settings, underscores its superior capability to generalize across challenging tasks, reinforcing its dominance in complex scenarios.

These insights not only augment our theoretical understanding of the dynamics between algorithmic strategies and hyperparameter settings but also provide valuable practical guidance for their application in a variety of contexts. The consistent finding that a higher initial epsilon generally enhances performance and learning smoothness across algorithms highlights the importance of exploration in the early stages of learning. By foregrounding the value of initial exploration, our findings offer a nuanced perspective on the design and implementation of reinforcement learning strategies, paving the way for future advancements in the field.

Conclusion

This investigation not only sheds light on the pivotal roles of hyperparameter tuning, reward shaping, and algorithm selection in solving complex RL tasks but also highlights Q-Learning's exceptional adaptability and generalization capabilities. While SARSA presented a volatile learning journey and Monte Carlo a rapid yet erratic path, Q-Learning's smooth and consistent progress stood out as a model of efficiency and robustness.

For further details, code, and data related to this research, please visit our GitHub repository at https://github.com/Lior-Baruch/RL or Google-Colab at https://colab.research.google.com/drive/1dkANLj-BSs6XCIi6bZrXvgvF629Snn2U?usp=sharing