Reinforcement Learning Individual Assignment

Maria Kontaratou

Centralesupélec, 3 Rue Joliot Curie, 91190 Gif-sur-Yvette, FRANCE

Abstract. This project explores the effectiveness of two reinforcement learning algorithms—Monte Carlo (MC) and SARSA(λ)—in solving the Text Flappy Bird (TFB) game environment. The objective was to compare their performance, convergence speed, sensitivity to hyperparameters, and adaptability across different environment configurations. Results indicate MC Control provides smoother but slower learning, whereas SARSA(λ) demonstrates faster convergence yet requires meticulous hyperparameter tuning. The implementation details and code for this project are available at: GitHub Repository.

1 Agent Implementation and Experimental Setup

This project implements two reinforcement learning agents—**Monte Carlo (MC)** and **SARSA(λ)**—to solve the TextFlappyBird-v0 environment, which provides a structured state representation based on the bird's distance to the next pipe gap. Both agents use ϵ-greedy policies for balancing exploration and exploitation, with Q-values stored in tabular format.

Monte Carlo Implementation For the Monte Carlo (MC) agent, I implemented an episodic learning approach using First-Visit MC Control with an ϵ-greedy policy. The `train_mc_agent` function maintains a Q-table, where state-action pairs are updated only at the end of an episode. Episodes are generated via `generate_mc_episode`, which selects actions based on `compute_action_probabilities`, prioritizing the best-known action while allowing exploration. Returns are computed using discounted rewards, and updates are performed with a constant step-size parameter (α). To balance exploration and exploitation, ϵ decays over time using `epsilon_decay`. This results in stable learning, as MC updates average returns over multiple episodes, leading to smooth policy convergence. However, MC is highly dependent on full episode completion, making it less effective in environments with frequent terminations or highly dynamic state spaces.

SARSA(λ) Implementation For SARSA(λ), I implemented a TD(λ) learning approach with eligibility traces, enabling faster learning by updating multiple state-action pairs per step. The `train_sarsa_lambda` function initializes Q-values and eligibility traces, which are updated incrementally based on TD error. Actions are selected using an ϵ-greedy strategy (`epsilon_greedy_action`),

with ϵ decaying over time. Unlike MC, SARSA(λ) updates the Q-table at each time step, incorporating bootstrapped estimates from subsequent states. The eligibility traces decay exponentially, ensuring that recent state-action pairs receive higher weight. This makes SARSA(λ) more adaptive to immediate rewards but introduces greater sensitivity to hyperparameters such as λ and step size (α). While SARSA(λ) converges faster, it exhibits higher variance compared to MC due to continuous updates and bootstrapping errors, requiring careful parameter tuning to stabilize learning.

For training, both agents were run for 100,000 episodes, tracking cumulative rewards, state-value functions, and learned policies. The Monte Carlo agent follows an episodic learning approach, updating Q-values only after completing full episodes using the First-Visit Monte Carlo method. In contrast, the SARSA(λ) agent updates Q-values incrementally at each step based on Temporal-Difference learning with eligibility traces, allowing for faster adaptation but introducing greater sensitivity to hyperparameters.

The experiment also involved hyperparameter tuning for key variables, including learning rate (α), discount factor (γ), exploration rate (ϵ), and eligibility trace decay (λ), to analyze their impact on convergence time and reward stability.

2 Performance of Agents: MC vs. SARSA(λ)

Policy Analysis The policy plots (Fig.2) highlight distinct decision-making strategies between the two agents. MC establishes a well-defined boundary between "Flap" (40%) and "Idle" (60%), indicating stable convergence. The agent primarily chooses to flap when $X-position < 5$ or during rapid descent (negative $Y-velocity$), suggesting a structured and predictable policy. In contrast, SARSA(λ) exhibits higher variability, with a more even flap-idle distribution, reflecting its responsiveness to immediate rewards but a lack of clear action boundaries. While this allows for quicker adaptations, it comes at the cost of policy stability compared to the more structured MC approach.

State-Value Function Analysis The state-value functions(Fig.3) highlight key differences in learning behavior between the two agents. MC exhibits smooth, gradual value updates, with values ranging from 0 to 35, reflecting its stable, episodic averaging. In contrast, SARSA(λ) shows more fluctuations and sharper peaks, reaching around 80 in certain regions, indicating its aggressive, incremental updates driven by eligibility traces. This contrast underscores MC's steady and risk-averse strategy, while SARSA(λ) adapts dynamically but is more sensitive to variations in rewards, making it more volatile but quicker to adjust. As a result, MC tends to learn more stable long-term policies, whereas SARSA(λ) may find optimal strategies faster but with less consistency.

Hyperparameter Tuning The hyperparameter tuning(Fig.4, Fig.5) results for Monte Carlo (MC) and SARSA(λ) agents highlight key differences in their learning behavior and sensitivity to parameter selection. The MC agent exhibits steady and consistent learning, with cumulative rewards gradually increasing under optimal hyperparameters, particularly when using moderate step sizes (e.g., 0.4) and lower epsilon decay, reaching peak rewards above 1400. This stability stems from its episodic updates, which average out variability and result in reliable long-term value estimates. In contrast, SARSA(λ) demonstrates faster initial learning, frequently achieving peaks around 700, but at the cost of significant volatility due to its stepwise updates. The fluctuating performance across different configurations indicates SARSA(λ)'s high sensitivity to step size and exploration rate, requiring careful tuning to balance learning speed and stability. While MC's stability makes it well-suited for environments where full episodes can be consistently completed, SARSA(λ) is more adaptable to changing conditions but requires precise hyperparameter adjustments to prevent instability.

3 TextFlappyBird-v0 environment

The two Text Flappy Bird environments differ in state representation: TextFlappyBird -v0 provides a simplified numerical state (X, Y distance to the next pipe), making it ideal for tabular reinforcement learning methods like Monte Carlo and SARSA(λ). In contrast, TextFlappyBird-screen-v0 uses pixel-based observations, drastically increasing complexity and making it better suited for deep learning approaches such as Convolutional Neural Networks (CNNs) or Deep Q-Networks (DQNs). Since this project focuses on tabular reinforcement learning, TextFlappyBird-v0 was chosen for its structured and manageable state space, as traditional methods struggle with high-dimensional inputs.

Applying the same Monte Carlo and SARSA(λ) agents to the original Flappy Bird environment would be challenging due to its continuous state space and high-dimensional pixel-based observations, which tabular RL methods struggle to handle efficiently. Storing Q-values for all possible states would lead to impractical memory usage, and the agents would require state discretization techniques, such as binning or tile coding, to function effectively. Even with these adjustments, deep reinforcement learning methods like DQNs or policy gradient approaches would be better suited for learning optimal policies in the original game.

4 New environments

Assessing the performance of the Monte Carlo (MC) and SARSA(λ) agents in different environment configurations helps evaluate their ability to generalize beyond training conditions. While both agents perform well in familiar settings, changes in height, width, and pipe gap affect their efficiency. This analysis explores how these variations impact their cumulative rewards, highlighting differences in adaptability and stability across configurations.

Height Sensitivity (Fig.6) Both agents demonstrate varying performance when tested on different heights. Monte Carlo (MC) maintains a steady reward level, showing minimal sensitivity to height changes. In contrast, SARSA(λ) shows gradual improvement as height increases, suggesting it benefits from greater vertical space. However, at extreme heights, SARSA(λ)'s performance fluctuates, possibly due to encountering less familiar state-action pairs. This indicates that while MC applies a more stable learned policy, SARSA(λ) adapts dynamically but may require more exposure to different height variations during training.

Width Sensitivity (Fig.7) Both agents generally benefit from increased width, as it provides more room for maneuvering. Initially, Monte Carlo performs slightly better, but as width increases beyond the training condition, SARSA(λ) shows stronger improvements. This suggests that SARSA(λ)'s adaptability helps it leverage the additional space more effectively. However, at extreme widths, both agents show variability, likely due to encountering new spatial patterns not present during training. These findings highlight that while MC retains consistent decision-making, SARSA(λ) is more responsive to environmental changes.

Pipe Gap Sensitivity (Fig.8) Monte Carlo (MC) remains highly stable across different pipe gaps, showing little variation in performance. SARSA(λ), on the other hand, exhibits some fluctuations, particularly with smaller gaps. This suggests that SARSA(λ) is more sensitive to changes in obstacle spacing, whereas MC maintains a consistent policy regardless of the gap size. The results indicate that MC's episodic learning approach helps it generalize better across varying pipe gaps, while SARSA(λ) may require more fine-tuned adjustments to optimize its performance in different conditions.

5 Figures

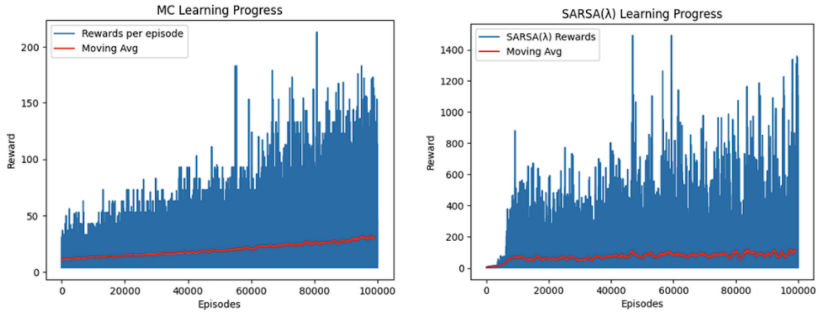

Fig. 1. Learning Progress of Monte Carlo (MC) and SARSA(λ) Agents

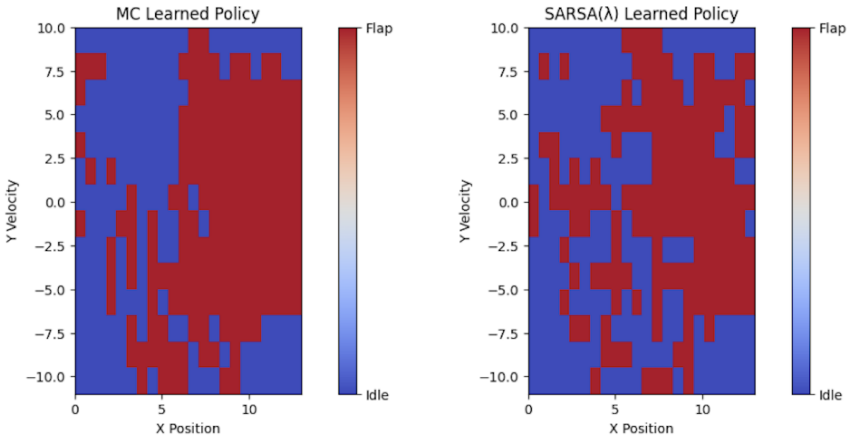

Fig. 2. Comparison of Learned Policies for MC and SARSA(λ) Agents

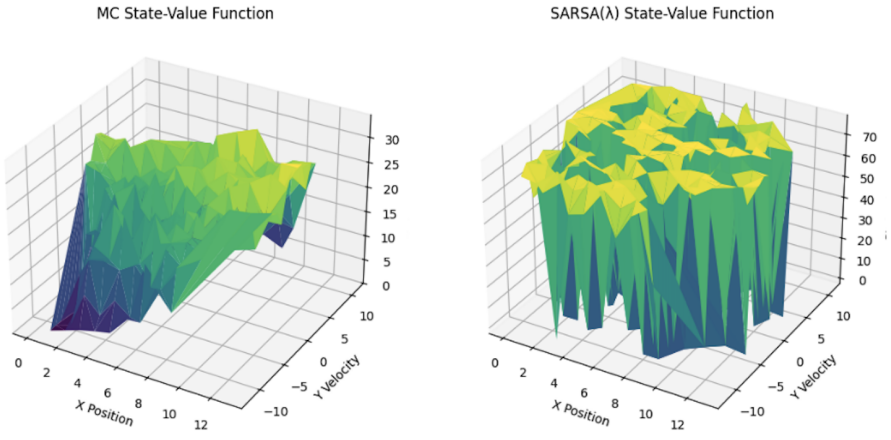

Fig. 3. Comparison of State-Value Functions for MC and SARSA(λ) Agents

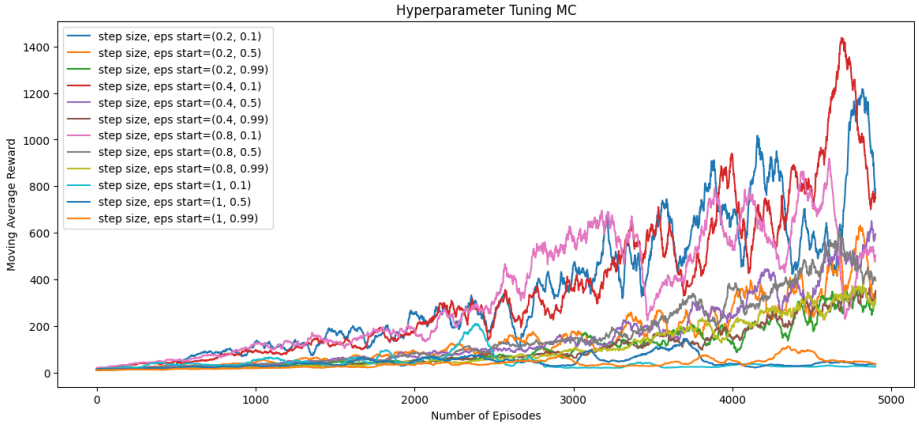

Fig. 4. Hyperparameter Tuning MC

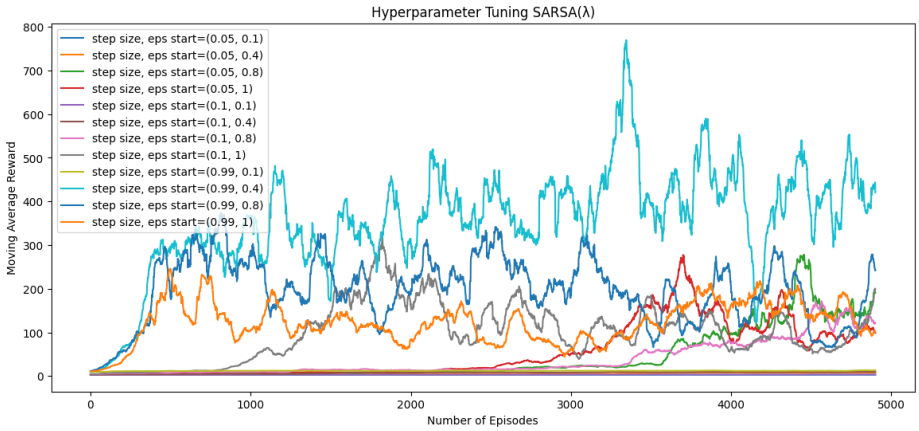

Fig. 5. Hyperparameter Tuning Sarsa(λ)

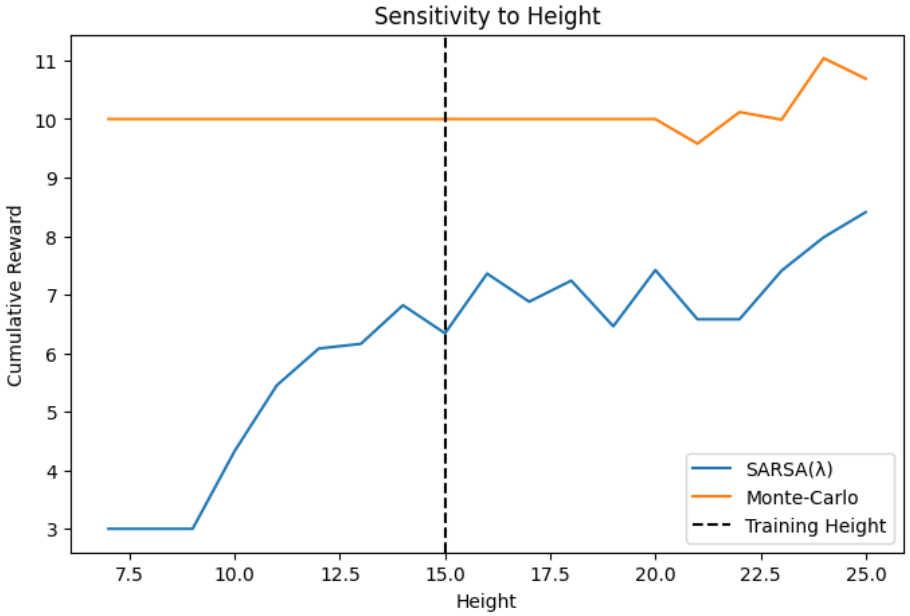

Fig. 6. Sensitivity Analysis of MC and SARSA(λ) Agents to Environment Height

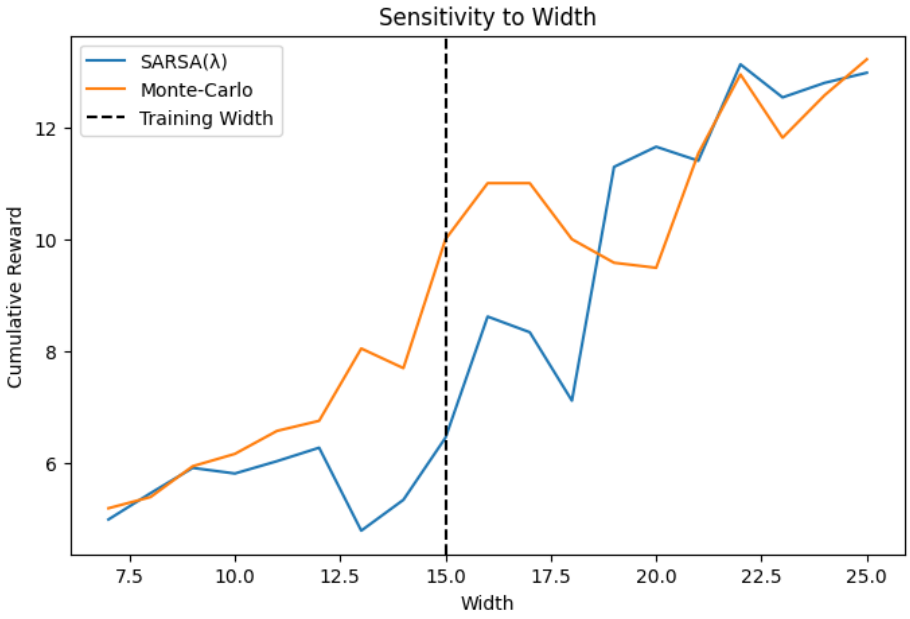

Fig. 7. Sensitivity Analysis of MC and SARSA(λ) Agents to Environment Width

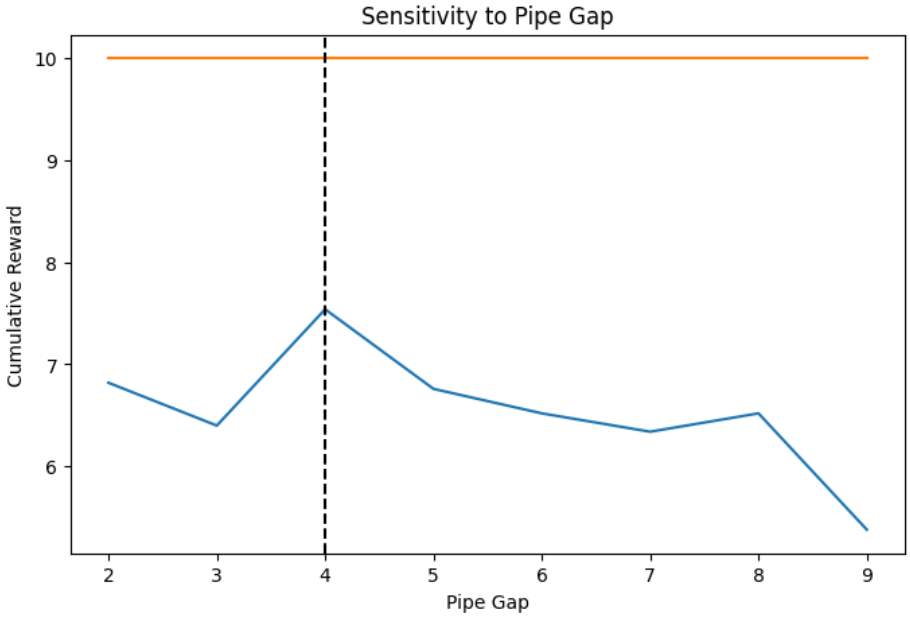

Fig. 8. Sensitivity Analysis of MC and SARSA(λ) Agents to Environment Pipe Gap