TFB Individual Assignment

Ayush Tankha (B00802762)

Ms Data Science and Business Analytics – ESSEC x CentraleSupelec

Github Link - *https://github.com/ayushtankha/RL---Text-Flappy-Bird-Project*

Abstract. To implement a monte-carlo and sarsa lambda agent for the provide text flappy bird environment. We also delve deeper into the analysis of rewards , state value function ,score and convergence of the agents.

Keywords: Sarsa-Lamda, Reinforcement Learning, Monte-Carlo

1 Initial Problem Set

1.1 Problem Introduction

As a part of the assignment for the subject of reinforcement learning, we have to apply namely two different agents – Monte-Carlo and Sarsa Lambda to play the game of Flappy Bird. We are also required to compare the two agents on the bases of scores, rewards, state value functions and convergence. We need to observe the agents sentitvity to parameters and find the best possible configuration for our agents

2 Execution

2.1 Agent Selection

The development of the Monte Carlo and SARSA (λ) agents leverages their individual reinforcement learning methods, emphasizing the acquisition of knowledge from ongoing interactions within each episode rather than waiting until the end. This approach to learning is crucial in games such as Flappy Bird, where episodes can be prolonged. Postponing learning until an episode's completion can be suboptimal, as highlighted by Sutton & Barto [1]. Both agents adopt an off policy learning style, refining their strategies through real-time experiences in the game. While predicting an ideal policy for Flappy Bird might be challenging, these agents are tailored to dynamically refine their approaches based on environmental feedback, despite the possibility of not developing exhaustive policies for all potential states. SARSA (λ) takes into account both immediate rewards and anticipatory future rewards for updating value estimates, whereas Monte Carlo focuses on the learnings from complete epi-

sodes. This fundamental difference could lead to notable contrasts in how they perform and learn through training.

2.2 Implementation

In the Monte Carlo agent implementation, an epsilon-greedy policy with an exploration rate (eps) of 0.1 guides action selection, balancing between exploration and exploitation. For each episode, rewards are backpropagated post-completion, applying a discount factor (gamma) of 0.99 to future rewards. The agent's Q-table, initialized to zero values for all state-action pairs, aggregates the returns for each unique state-action occurrence within an episode. This aggregation forms the basis for the Q-value updates, leading to an evolving policy that aspires to maximize cumulative rewards. Over many episodes, the agent refines its Q-table, using it to derive a state-value function (v_table) that underpins the decision-making process, selecting the action with the highest expected return. Hypertuning is also done in this to get the best parameters (done on 10,000 episodes)

The SARSA (λ) agent employs a similar epsilon-greedy approach but introduces a learning rate (alpha) of 0.5 and an eligibility trace decay factor (lambd) of 0.8, providing a more dynamic and temporally sensitive update mechanism. After each action is taken, the agent adjusts the Q-values for the current state-action pair, as well as for all preceding pairs in the episode, scaled by their eligibility trace. This method captures the temporal difference (TD) error — the discrepancy between predicted and realized rewards — to inform immediate and successive Q-value adjustments. This TD error informs not just the update of the current state-action value but propagates through earlier visited states, fading as the traces diminish, hence implementing a more complex feedback loop for learning from both immediate outcomes and anticipated future states. A point to be noted that is that these parameters are chosen after hyper parameter tuning as seen in *Figure 3.* The hyper tuning in this case is done over 500 episodes due to memory constraints.

3 Model Evaluation and Comparison

Three primary elements were compared between the SARSA (λ) and Monte Carlo agents in the Flappy Bird gaming environment: generalization ability, learning progression as shown by rewards and scores over training episodes, and state-value functions.

3.1 Training and Convergence time

Using our observations from *Figure 2,* during training, rewards and scores captured in Monte Carlo agent can be seen to gain sudden increase after 4000 episodes. This also shows a positive trend in variability in rewards indicating in exploration and convergence of agent algorithm. After hypertuning and observing the reward and score functions for the Sarsa Lambd we also observe a positive trend. This convergence trend

however is lower than Monte Carlo but starts from the initial 0 episodes, which shoes the temporal nature of the agent.

3.2 State Value Function

As seen in *Figure 1* the state value function for Monte Carlo has been normalized using log-normal to make it look more appealing to the user. Here we can observe states of exploration and exploitation in reference to the color coding of the heatmap. Brighter sells indicate a favorable outcome (like passing a pipe) while darker cells indicates unfavorable situations. The discretized nature of the heatmap and the varying intensities of the colors mirror the stochastic nature of the agent's exploration of the game space, with an epsilon parameter (ε) that governs the trade-off between exploration and exploitation. For sarsa lambd. The x and y axes correspond to discretized game states, which are factors such as the bird's altitude or horizontal distance from an obstacle, and the z-axis represents the estimated value of being in that state. The undulating nature of the graph reflects the temporal-difference learning characteristic of SARSA (λ), which accounts for the value of both the present action and those that follow, up to a lookahead determined by the decay parameter λ.

3.3 Result Comparison

When contrasting the two agents, it appears that the SARSA (λ) agent acquires knowledge at a quicker pace, yet with more variability in its rewards and scores. This suggests a policy that is more inclined towards exploration but may lack consistency. Conversely, the Monte Carlo agent, which bases its learning on full episode outcomes, displays steadier progression and seems to develop a more stable policy, leading to performance with less fluctuation. Analyzing the behavior of each agent's learning through their respective graphs highlights their unique approaches to learning and policy development, shedding light on the benefits and drawbacks of learning incrementally versus relying on complete sequences of gameplay. The methodical approach of the Monte Carlo agent stands in contrast to the SARSA (λ) agent's responsive policy refinement, each offering its advantages within the context of the Flappy Bird game dynamics.

4 Agent use in complex environements

Training agents in 'TextFlappyBird-screen-v0' involves a state space defined by distances to the pipe gap, yet the environment's screen array output is computationally demanding for direct use. A practical approach is to extract key distances or abstract the state space to simplify it, requiring adjustments to the agents' design. For the original Flappy Bird setup with continuous observations, current agents are unsuitable due

to memory constraints and the need for handling continuous inputs, pointing to the necessity for agent's adept at processing or approximating continuous data.

5 References

1. R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction, Cambridge. 2nd edn. The MIT Press, Cambridge MA (2018).

6 Appendix

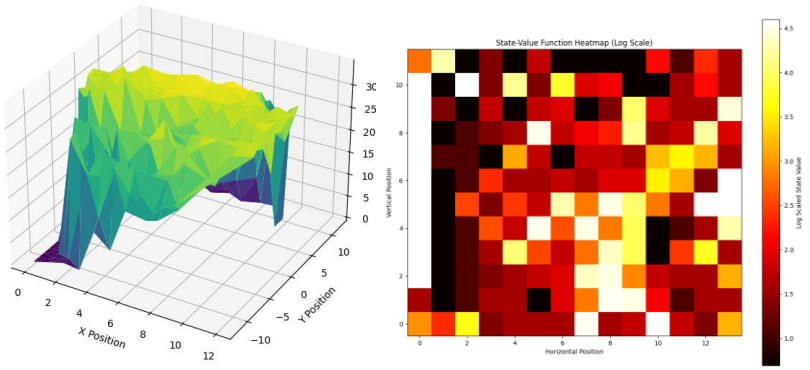
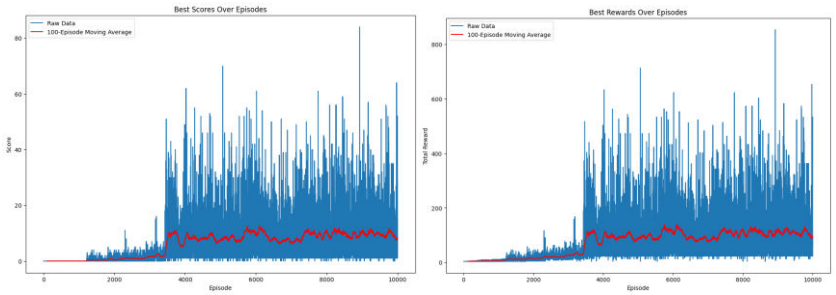


Figure 1 - State Value Function Plots Sarsa(λ) vs Monte Carlo



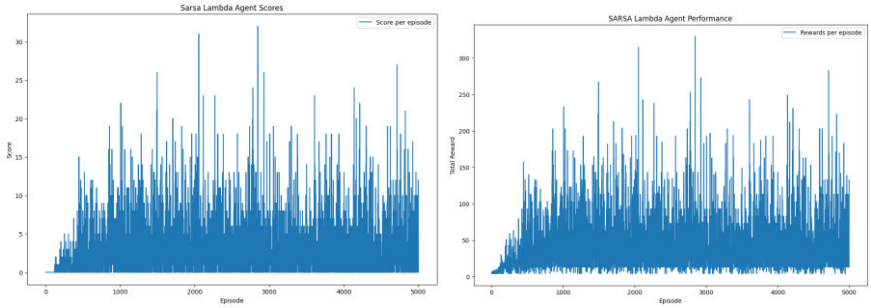


Figure 2 – Optimal Performance Model

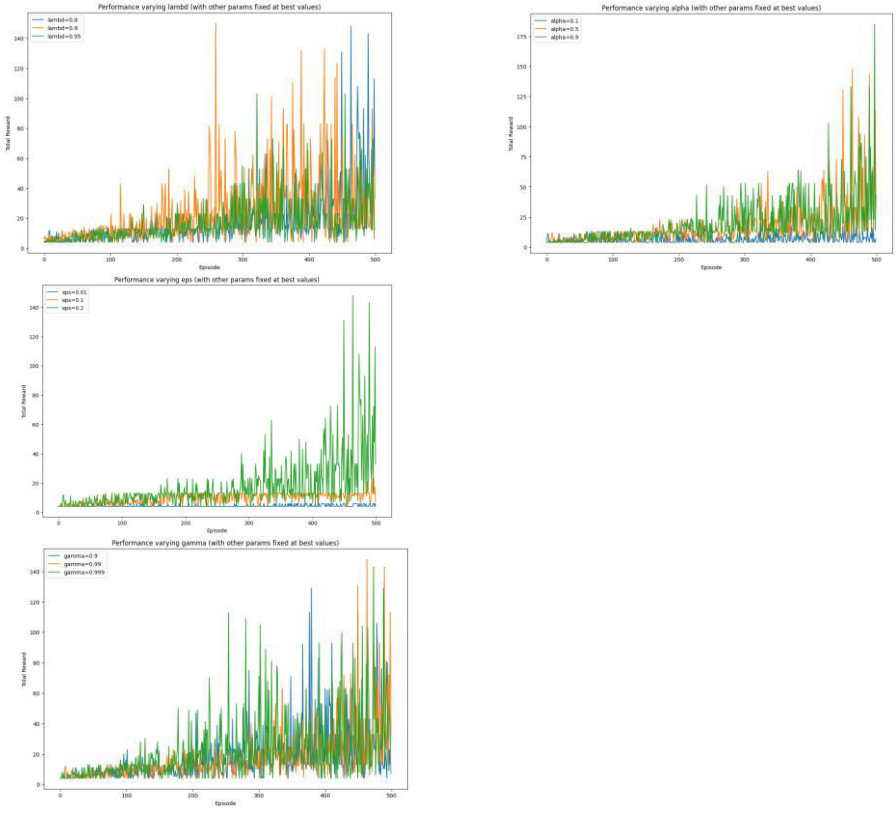


Figure 3 – Sarsa(λ) hypertuning parameters