Apply REINFORCE and SARSA (λ) Agents to Learn Text Flappy Bird Game

Zhongke SUN[1]

CentraleSupélec, Gif-sur-Yvette 91190, France

Colab Page: Here is the hyperlink to Colab Python Notebook

1 Flappy Bird & Text Flappy Bird

Text Flappy Bird is a variation to a well-known game - Flappy Bird, in which the player is made with a simple unit-element character. The goal is to avoid different walls and get the score as high as possible.

The player can choose two actions: 'Flap up' and 'Flap down' to avoid the obstacles. In this assignment, I will train a Monte Carlo based agent (REINFORCE agent) and the SARSA (λ) to play Text Flappy Bird.[2]

1.1 How are the two versions of the TFB environment different?

TextFlappyBird-screen-v0

- Returns the complete screen render of the game as an observation. The observation is an array representing the current state of the game screen encoded as integers.
- Provides a raw visual representation similar to what a human player would see.

TextFlappyBird-v0

- Returns only the distance of the player from the center of the closet upcoming pipe gap. The observation consists of just two values: the distances along the x and y axes.
- Provides a highly abstracted, simplified representation of the game state.

1.2 What are the main limitations of using one or the other environment?

TextFlappyBird-screen-v0 (Screen-based) Limitations

- Much higher dimensional state space, making learning more complex and potentially slower.
- May need preprocessing to convert the integer-encoded screen to a more useful representation.

TextFlappyBird-v0 (Distance-based) Limitations

- Discards potentially useful information about the game state. May not provide enough context about upcoming obstacles beyond the closest pipe.
- Relies on the environment doing the feature extraction, making it less generalizable.

1.3 With an implementation of the original Flappy Bird game environment available, can the same agents be used?

The same REINFORCE and SARSA (λ) agents can be available with some modifications. The *TextFlappyBird-v0* is a simplified, text-based version but *FlappyBird-v0* is a graphical environment with continuous state representation.

State Representation FlappyBird-v0 provides a 4-dimensional state space: $[player\_y, player\_velocity, next\_pipe\_x, next\_pipe\_y]$. But current agents are expecting a 2D state structure, so they need to be modified to process 4D state vectors.[3]

Action Space Both environments share the same action space: $[0 : \mathit{Flap\ down}, 1 : \mathit{Flap\ up}]$. Hence, no changes are needed in action handling.

2 Two agents and their implementation experiments

2.1 Monte Carlo based agent - REINFORCE

The REINFORCE algorithm [1] is a Monte Carlo variation of policy gradient algorithm in RL. The agent collects the trajectory of an episode from current policy.

2.2 SARSA (λ) agent

SARSA (State-Action-Reward-State-Action) is an on-policy reinforcement learning algorithm that updates Q-value based on the agent's actual experiences, including exploratory actions. [4]

2.3 Experimental Setup

Here I will do investigation on how the two implemented agents differ. I will conduct experiments based on 4 features:

- **Sensitivity to parameters**: Normalized Sensitivity Coefficient (NSC)[5].

$$\text{NSC} = \frac{\Delta R}{\Delta P} \times \frac{P_{\text{mean}}}{R_{\text{mean}}} \tag{1}$$

- **Convergence time**: The first episode where the moving average of the last *window_ size* episodes is greater than or equal to *threshold*.
- **State-Value Function**: State-value function $V(s)$ represents the expected return (cumulative future rewards) when starting from a given state s and following a specific policy π.
- **Scores**: The score is the average smoothed score in *window_ size = 50*.

Environment I used $env = gym.make('TextFlappyBird{-}v0', height = 15, width = 20, pipe\_gap = 4)$. For the experimental episodes setting, I set 1000 for REINFORCE agent and 1500 for SARSA agent, print the scores each 250 episodes. (Because there are a lot of parameters to test, 1000 episodes are enough for REINFORCE agent to converge and determine the optimal parameters. But SARSA agent needs 500 episodes more).

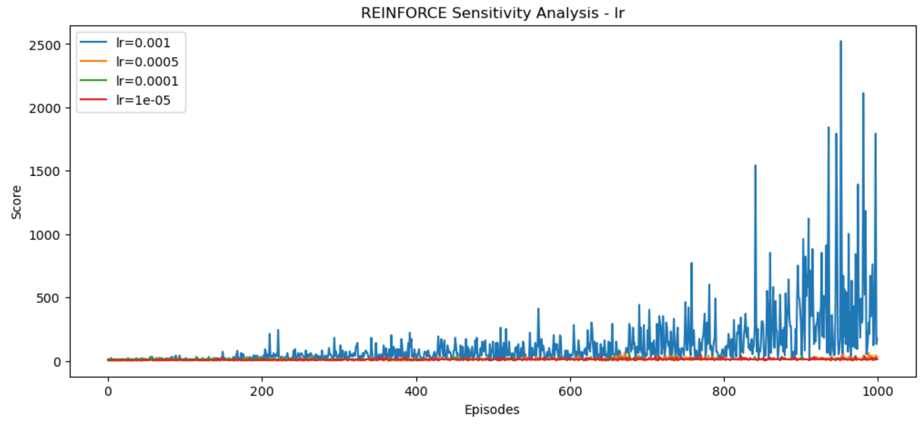
Base Parameters & Parameters Grid I conducted experiments on each variable to determine the optimal parameter values for the agent. The goal is to identify the best hyperparameters settings by evaluating the corresponding optimal reward, scores, sensitivity, and convergence time. The table below demonstrates the baseline setting and the parameter search space.

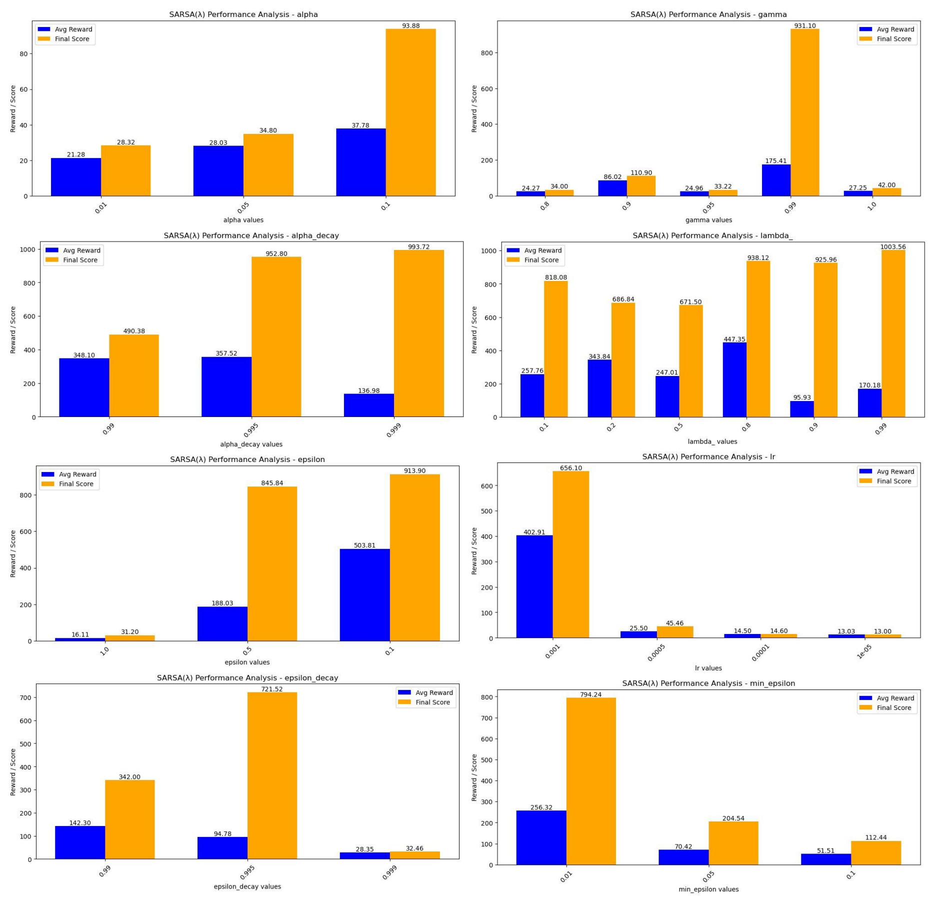
Algorithm	Parameters	**Base Value**	**Parameter Grid**
REINFORCE	Gamma (γ)	0.99	{0.8, 0.9, 0.95, 0.99, 1.0}
	Learning Rate (α)	5e-4	{1e-3, 5e-4, 1e-4, 1e-5}
SARSA(λ)	Gamma (γ)	0.99	{0.8, 0.9, 0.95, 0.99, 1.0}
	Lambda (λ)	0.9	{0.1, 0.2, 0.5, 0.8, 0.9, 0.99}
	Epsilon (ϵ)	1.0	{1.0, 0.5, 0.1}
	Epsilon Decay (ϵ_{decay})	0.997	{0.99, 0.995, 0.999}
	Min Epsilon (ϵ_{min})	0.01	{0.01, 0.05, 0.1}
	Alpha (α)	0.1	{0.01, 0.05, 0.1}
	Alpha Decay (α_{decay})	0.995	{0.99, 0.995, 0.999}
	Learning Rate (α)	5e-4	{1e-3, 5e-4, 1e-4, 1e-5}

Table 1. Base Parameters and Parameter Grid for REINFORCE and SARSA(λ)

2.4 Best Hyperparameters

I utilized training log, line-plot and bar-plot (see below) to determine the best parameters. You will see the best parameters experimented as the following table (see Table 2.).


REINFORCE Sensitivity Analysis - lr
lr=0.001
lr=0.0005
lr=0.0001
lr=1e-05
Score
Episodes


SARSA(λ) Performance Analysis - alpha
Avg Reward
Final Score
Reward / Score
alpha values
21.28
28.32
28.03
34.80
37.78
93.88
0.01
0.05
0.1
SARSA(λ) Performance Analysis - gamma
gamma values
24.27
34.00
86.02
110.90
24.96
33.22
175.41
931.10
27.25
42.00
0.8
0.9
0.95
0.99
1.0
SARSA(λ) Performance Analysis - alpha_decay
alpha_decay values
348.10
490.38
357.52
952.80
136.98
993.72
0.99
0.995
0.999
SARSA(λ) Performance Analysis - lambda_
lambda_ values
257.76
818.08
343.84
686.84
247.01
671.50
447.35
938.12
95.93
925.96
170.18
1003.56
0.1
0.2
0.5
0.8
0.9
0.99
SARSA(λ) Performance Analysis - epsilon
epsilon values
16.11
31.20
188.03
845.84
503.81
913.90
1.0
0.5
0.1
SARSA(λ) Performance Analysis - lr
lr values
402.91
656.10
25.50
45.46
14.50
14.60
13.03
13.00
0.001
0.0005
0.0001
1e-05
SARSA(λ) Performance Analysis - epsilon_decay
epsilon_decay values
142.30
342.00
94.78
721.52
28.35
32.46
0.99
0.995
0.999
SARSA(λ) Performance Analysis - min_epsilon
min_epsilon values
256.32
794.24
70.42
204.54
51.51
112.44
0.01
0.05
0.1

Algorithm	**Parameter**	**Optimal Value**	**Best Score**
REINFORCE	Gamma	0.8	3743.80
	Learning Rate	0.001	5136.60
SARSA(λ)	Gamma	0.99	931.10
	Lambda	0.99	1003.56
	Epsilon	0.1	913.90
	Epsilon Decay	0.995	721.52
	Minimum Epsilon	0.01	794.24
	Alpha	0.1	93.88
	Alpha Decay	0.999	993.72
	Learning Rate	0.001	656.10

Table 2. Best Parameters and Corresponding Scores for REINFORCE and SARSA(λ)

2.5 Comparisons

After determining the best hyperparameters, we can compare the 4 features of the two agents.

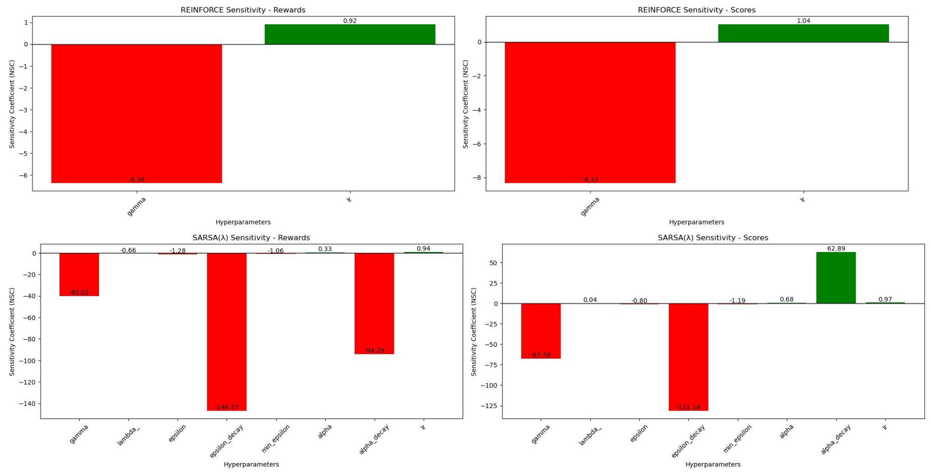


Fig. 1. REINFORCE & SARSA Parameters Sensitivities

Sensitivity to parameters The sensitivity analysis of two agents (Figure 1) reveals distinct hyperparameter influences on rewards and scores.

In REINFORCE, gamma shows strong negative sensitivity (-6.36 for rewards, -8.32 for scores), suggesting that higher discount factors hinder performance. In contrast, learning rate (lr) positively impacts both metrics (0.92 and 1.04), highlighting its importance in optimization.

SARSA(λ) exhibits extreme sensitivity to epsilon decay (-146.77 for rewards, -131.18 for scores), indicating that excessive decay disrupts learning. Gamma and epsilon also show negative effects, while alpha decay (62.89) and lr positively influence scores. Overall, REINFORCE is more stable, whereas SARSA(λ) is highly sensitive to exploration-related parameters.

Convergence Time After running the two agents with their optimal parameters for 750 episodes (excluding REINFORCE's gamma = 0.95, for faster possible convergence). I calculated the convergence time by finding the first episode where the moving average of the last *window_ size* episodes is greater than or equal to *threshold*.

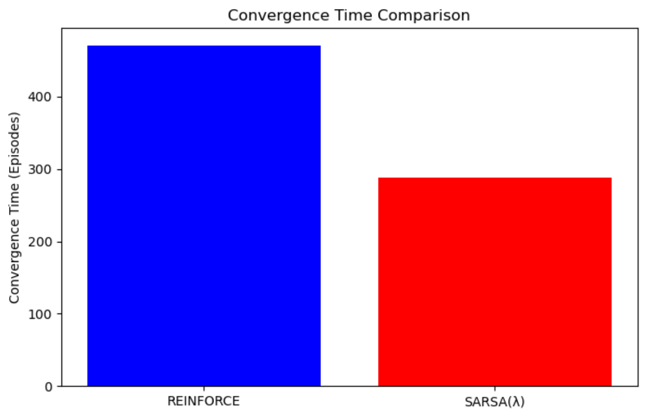


Fig. 2. Comparison of Convergence Time between REINFORCE and SARSA(λ)

The experimental result demonstrates that SARSA (λ) converges faster than REINFORCE(290 episodes v.s. 480 episodes). It's because SARSA agent use temporal difference method so it can quickly adapt Q value and converge. REINFORCE agent need to wait the entire episode finished then update policy, so it's slower relatively.

Scores The performance comparison between REINFORCE and SARSA(λ) reveals distinct learning dynamics. As shown in Figure 3, REINFORCE (blue) exhibits higher variance but achieves significantly higher scores in later episodes, indicating its strong capability for long-term reward optimization. In contrast, SARSA(λ) (red) maintains more stable but lower scores, reflecting its on-policy nature, which prioritizes safer exploration but may converge to suboptimal policies. But SARSA agent reaches a base score (like 3,000) earlier than REINFORCE agent. These results suggest that while REINFORCE benefits from its unbiased Monte Carlo updates, SARSA(λ) is more conservative and less prone to high reward fluctuations.

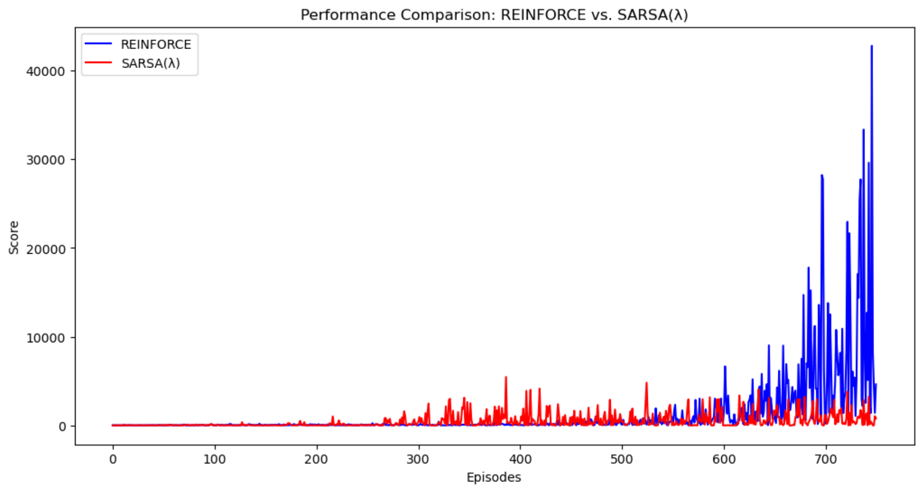
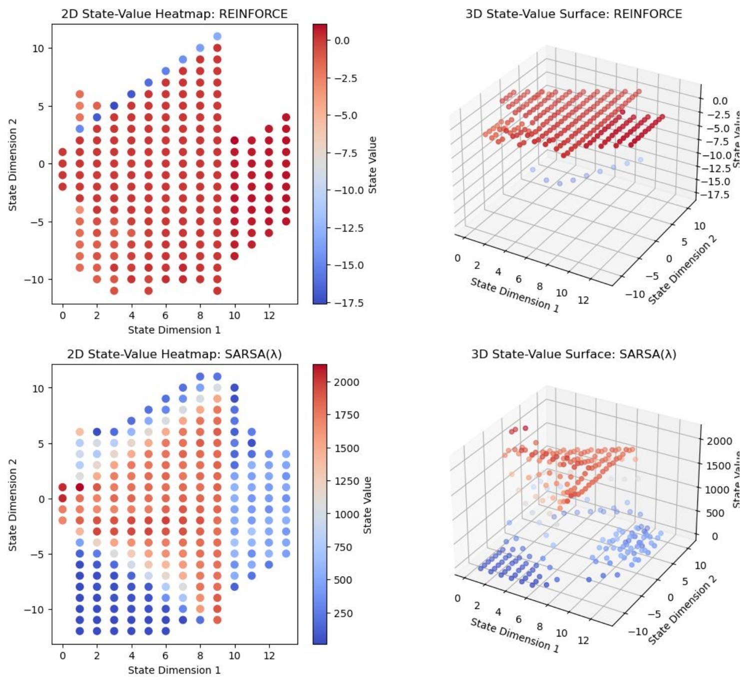


Fig. 3. SARSA & REINFORCE Score Comparison



State-Value Function From the state-value function comparison, we observe that REINFORCE and SARSA(λ) exhibit distinct state-value distributions. In REINFORCE, the state values are relatively smooth and consistently distributed, with values ranging from negative to near zero. This suggests that the policy-based method learns a more structured global estimation of state values, albeit with potentially high variance due to its Monte Carlo nature. In contrast, SARSA(λ) demonstrates a wider spread of state values, including significantly higher positive values, indicating that it is able to capture more granular local state estimations. This difference is likely due to the bootstrapping nature of SARSA(λ), which allows for incremental updates and more adaptive learning in different regions of the state space. However, SARSA(λ)'s state values also show greater fluctuation, possibly due to its reliance on the off-policy temporal difference (TD) updates, which can introduce instability in certain state regions. Overall, REINFORCE maintains smoother, lower-magnitude state values, while SARSA(λ) achieves a more diverse but less stable state-value distribution.

3 The impact of different configurations of the TFB environment

In this experiment, I also conducted an experiment to investigate how well the trained agent performs on a different level configuration of TFB environment. I tried:

1. **gym.make('TextFlappyBird-v0', height=10, width=15, pipe_gap=2)**
   - Increased difficulty: The smaller height and width provide less room for maneuvering, while a pipe gap of 2 makes navigation more challenging. The agent must make quicker and more accurate decisions to avoid collisions.
   - My agent was trained in a larger environment, so it struggled a bit to adapt, leading to relatively lower scores.
2. **gym.make('TextFlappyBird-v0', height=20, width=25, pipe_gap=6)**
   - Lower difficulty: The increased height and width provide more space for movement, and a pipe gap of 6 allows for easier passage.
   - The agent achieved higher scores, as mistakes are less likely to result in immediate failure.
   - Possible overcompensation issues: If the agent was trained to make frequent adjustments in a more constrained space, it might initially overreact to obstacles before adjusting to the larger environment.

References

1. Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3-4), 229–256. https://doi.org/10.1007/BF00992696
2. Text Flappy Bird for OpenAI Gym, `https://gitlab-research.centralesupelec.fr/stergios.christodoulidis/text-flappy-bird-gym`, last accessed 2025/03/19
3. Flappy Bird for OpenAI Gym, `https://github.com/Talendar/flappy-bird-gym?tab=readme-ov-file`, last accessed 2025/03/19
4. Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. 2nd ed., MIT Press, 2018.
5. Sensitivity Methods in Control Theory. Pergamon, 1966. https://doi.org/10.1016/C2013-0-07907-8