Reinforcement Learning - Individual Assignment

Paul Bédier

MSc Data Science & Business Analytics, CentraleSupélec

Abstract. For this assignment, the goal was to implement two agents to try to solve the Text Flappy Bird Environment provided at github. I chose to implement a Q-Learning agent as well as a SARSA agent, and managed to train both of them to beat the game and yield potentially infinite scores. The python notebook containing implementations and training can be found at my github

1 Chosen Agents - Rationale

1.1 Q-Learning Agent

Q-Learning is the first agent I decided to implement as I believe, on top of its simplicity, it is well-suited to the task of solving the Flappy Bird Game environment as the following reasons explain:

1. Simple state space: The state space of the Flappy Bird game is relatively simple, consisting of the bird's distance from the closest upcoming pipe gap. This makes it easy to represent the state space as a table of Q-values.
2. Discrete actions: The available actions in Flappy Bird are discrete (flap or not flap), which is a good fit for Q-learning, as it can learn the optimal action-value function for each state-action pair.
3. Exploration: Since Flappy Bird is a stochastic game with random obstacles, exploration is crucial for finding optimal policies. Q-learning's epsilon-greedy exploration strategy can help the agent to explore the state space effectively.

1.2 SARSA Agent

SARSA is the second agent I decided to implement in order to try and improve on the previous agent. Being mostly based on the same framework as Q-Learning, it is well-suited for the same reasons as above. However, the main difference with Q-Learning lays with the off-policy / on-policy learning trade-off. SARSA uses the latter, meaning that it is learning the same policy it is using to select actions (this is not true for off-policy Q-Learning). This can help solving our environment, especially since exploration will be critical and on-policy learning could help the learning agent converge faster despite this.

2 Learning Phase - Hyperparameter Tuning

For both agents, I performed training runs and grid search / parameter sweeps over possible values:

2.1 Q-Learning Agent

Table 1: Q-Learning Parameters

Parameter	Grid Range	Retained	Comment
Step-size	0.1 - 0.9	0.40	Highest performance, balance between past/present experiences
Epsilon	0.001 - 0.2	0.025	Enforced lower exploration to help convergence
Discount	1 - 0.65	1	Biased towards long-term rewards
Step-size Decay	1 - 0.85	0.99	A bit of decay to help convergence
Epsilon Decay	1 - 0.85	0.99	Decay otherwise slow convergence under exploration

2.2 SARSA Agent

Table 2: SARSA Parameters

Parameter	Grid	Retained	Comment
Step-size	0.1 - 0.9	0.50	Highest performance, balance between past/present experiences
Epsilon	0.001 - 0.2	0.05	Enforced higher exploration to help performance
Discount	1 - 0.65	1	Biased towards long-term rewards
Step-size Decay	1 - 0.85	0.99	A bit of decay to help convergence
Epsilon Decay	1 - 0.85	1	No decay as convergence efficient under exploration

3 Agent Comparison & Results

Let's have a closer look at how the agents implementation differ beyond parameter choice.

3.1 Convergence & Sensitivity

Convergence time is a significant factor in the comparison between these two agents. As mentioned before, exploration is important to the task of solving the Flappy Bird environment, however high epsilon values can lead to sub-optimal convergence times. This is especially true of Q-Learning, as its off-policy learning means it can learn different policies which is good in principle for exploration but also means the convergence takes longer. This is why I selected a moderate epsilon value for Q-Learning, and a faster epsilon decay rate, in order to fasten convergence.

On the other hand, SARSA's off-policy learning means it can tackle this trade-off between exploration and convergence by learning the same policy it uses for action selection. This is why I was able to pick a higher epsilon value, and use a lower epsilon decay rate. Below is a graph comparing convergence for both final versions of the agents (SARSA has lower amount of overall runs as the 1M runs on Q-learning was not necessary, so scale is different). Thanks to parameter tuning, both agents converge at around 5000 runs. This implies that witht the same parameters, Q-Learning would have probably converged later than SARSA.

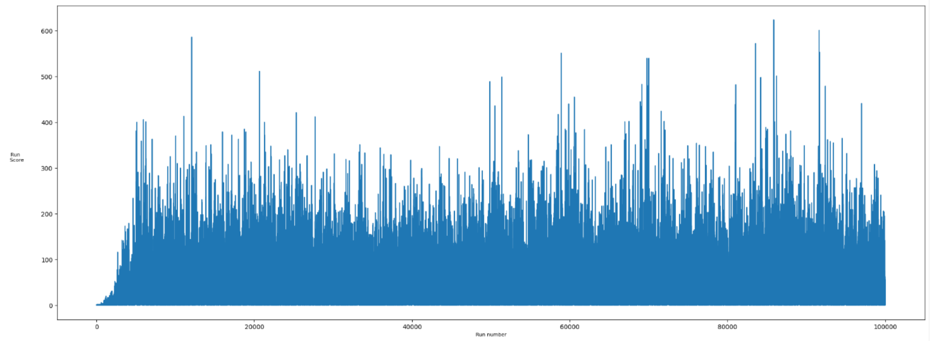

Fig. 1: Q-Learning Convergence

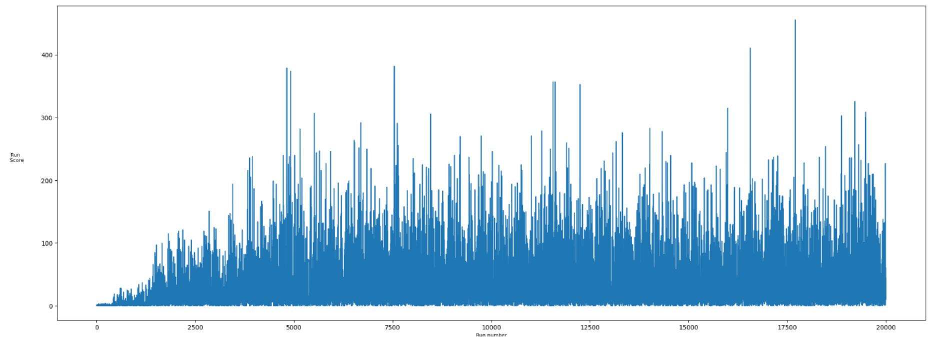

Fig. 2: SARSA Convergence

Both agents are extremely sensitive to step-size and discount parameters. A very low or very high step-size was, for both agents, detrimental to the performance. As it is too high, both agents overemphasize on recent experience and fail to converge to the optimal policy. As it is too low, both agents do not learn enough, leading to the same result. A sweet spot was found around 0.4-0.5, in both cases, with slight step-size decay. Anything else would harm performance.

With regards to discount, the Flappy Bird environment requires a quite large value as the ultimate goal is staying alive as long as possible. Low-value discount (i.e. large discount) would bias the agent towards pure obstacle avoidance and decrease long-term success: indeed, the best trajectory to take involves not only the upcoming pipe but the next ones as well. As such, agents are quite sensitive to discount value, and no discounts (i.e. value of 1) were used.

3.2 State-Value Functions

The State-Value function represents the expected cumulative reward that an agent can receive from a particular state onwards, under a given policy. By examining the state value function, one can gain insight into which states are considered more valuable by the agent and how the agent prioritizes exploration versus exploitation.

Below are the State-Value functions of both final versions of Q-Learning and SARSA:

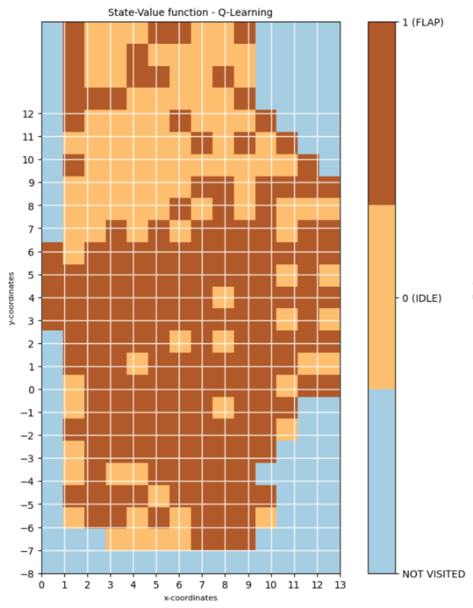

Fig. 3: Q-Learning

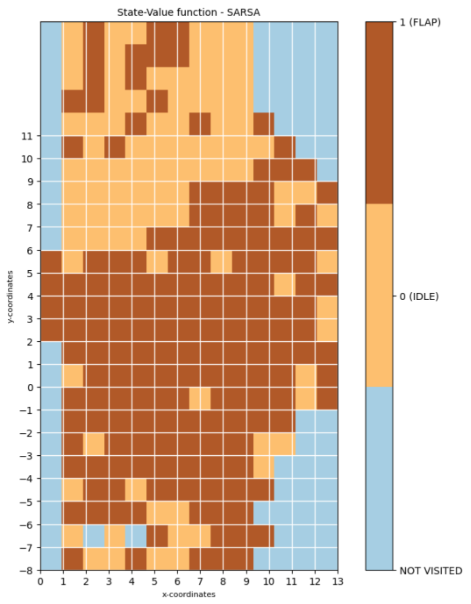

Fig. 4: SARSA

Overall, we find what we expect to see: the agent flaps most often when it is in the bottom part of the screen to get higher up, and stays idle in the upper part of the screen to fall back down. Most states have been explored, except near the corners. We do see a higher exploration rate for SARSA, with more states visited (e.g.: the bottom line).

3.3 Results

The final results obtained using optimal Q Tables obtained after training only (or sampling action when unvisited state appears), over 3 runs to eliminate random luck, are summarized in the table below:

Table 3: Results

Agent	Best Score	Best Reward
Q-Learning	Infinite (1M tested)	Infinite
SARSA	Infinite (1M tested)	Infinite

As such, both Q-Learning and SARSA manage to survive in Text Flappy Bird for what seems to be an infinite amount of time, yielding infinite scores/rewards. This is under my implementations and with the current parameters in used.

4 Further Discussion

The two provided environments differ most significantly in their state representation. *TextFlappyBird-v0* is a simple environment returning a 2D vector containing the distance to the closest upcomming pipe gap. Meanwhile, the *TextFlappyBird-screen-v0* environment is much more complex and returns the screen's full pixel array.

As such, one of the main limitation of using the same agents on this second more complex environment is that our current agents was not exposed to the full range of states and features present in this environment, and may therefore not generalize well. Another limitation is that the learned policy may simply be sup-optimal in this new environment.

Likewise, the original implementation of Flappy Bird would provide a different challenge. Such an environment is returning even wider state representations and features in both screen and standard formats. Our agents would fail to generalize for the same reasons as above.

References

1. Watkins, C.J.C.H., Dayan, P.: Q-learning In: Mach. Learn. vol. 8, no. 3, pp. 279–292, 1992. https://doi.org/10.1007/BF00992698
2. Rummery, G.A., Niranjan, M.: MCQ-L In: On-Line Q-Learning Using Connectionist Systems, 1994.