A Flappy Agent - 3MD4120 RL Assignment

Karim EL HAGE[1]

[1] CentraleSupélec Paris-Saclay, 3 rue Joliot-Curie, F-91192, Gif-sur-Yvette, FRANCE
karim.elhage@student-cs.fr

1 Problem Definition

1.1 Problem Introduction and Definition

As part of an assignment for 3MD4120 Reinforcement Learning, it is to apply Reinforcement Learning methods in a Flappy Bird game environment by training two agents to play the game of Flappy Bird. The task further requires the comparison of the two agents through discussion of their respect state-value functions, policy, and raw performance as well as other relevant comparison metrics in the context of the Flappy Bird environment. The agents shall be tested in an environment called *TextFlappyBird-v0 which* is a simplified version of the real environment by outputting the distance of the agent from the next pipe. However, the analysis should also allow for the projection of performance in the *TextFlappyBird-screen-v0*, which outputs a complete screen render of the observation. An explanation of the problem in the context of the environments, states, rewards, and agents can be found in the Main_Assignment_Code.ipynb at https://github.com/karimelhage/flappy-bird-agent-RL.git.

2 Implementation

2.1 Agent Selection

The two agents were modelled after the Q-Learning and Expected SARSA algorithms consequently. This was to leverage their Temporal Difference property of learning during the running of the episodes rather than at the end of each episode. This was found to be especially important considering that Sutton & Barto suggest that in applications where episode runtime can be very long, as could be the case with Flappy Bird, delaying all the learning till the termination of the episode is too slow [1]. By implementing Temporal Difference methods, the agent learns actively at each step within the episode. Furthermore, these algorithms learn off-policy allowing them to learn actively from the environment. Whilst a policy could be defined in the context of flappy bird, defining an optimal one could be complex and tedious. It could be more interesting to have the agent learn a policy actively from the environment. The one disadvantage here is that Flappy Bird might not have a comprehensive policy for all possible states but only those it explored. The main difference between Q-Learning and Expected SARSA is that a Q-Learning agent would take the maximum state-action value pair whereas the latter considers the likelihood of each action occurring under the current policy [1]. Hence, it would be interesting to observe the differences that this adjustment in an otherwise similar algorithm could have on training.

2.2 Notes on Implementation

Since the agent could theoretically run forever, a maximum score was set during training to kill the agent otherwise training would be unfeasible. In the beginning, hyperparameters were defined arbitrarily based on values seen in the lab to simply get a gauge on the problem. Originally, both agents were implemented with a fixed epsilon, but this was found to cause the agent to explore to not maximize performance after training. Hence, the algorithms were modified to later follow an epsilon decaying algorithm so that once the agent nears an episode where it reaches a close to the optimal policy, it begins to exploit the greedy action from its learnt policy. After that, hyperparameter tuning was conducted to test different pairs of step sizes and starting epsilon and find the optimal parameters for each algorithm that could maximize convergence time and performance. Other parameters, such as the number of episodes, were kept constant to be able to conduct a controlled comparison between the agents. Due to the computational time of this exercise, especially once an agent begins to converge, the maximum score was limited to 1500 after which the agent dies. Furthermore, the number of episodes was restricted to 10,000 per run. Training on optimal parameters was then done on a max score of 7,000 and 15,000 episodes.

3 MODEL EVALUATION AND COMPARISON

3.1 Comparison of Results between Q-Learning agent and Expected SARSA

Fig. 1 in the Appendix is a summary of the results when comparing the state-value function of the QL agent to that of the Expected SARSA agent. By taking the max value of q at each state, it can be observed that both agents are attempting to learn an optimal state-function function that once the dx (the x distance from the pipe center) tends to zero, there are limited dy (y distance from the pipe center) of which the agent would be rewarded. The exception to this rule is when the dx >= 10 at which point it is likely the agent is still crossing the previous pipe and needs to ensure not to crash into the previous pipe. The Q-Learning agent seems to have a smoother value function than that of the expected SARSA at this location or is perhaps an indication of conservatism from the latter agent. Furthermore, the latter seems to take a more conservative approach once the pipe agent is a dx <= 5 by outputting higher valuer for a smaller range of dy. **Fig. 2** in the Appendix highlights how the Q- Learning agent has tended closer to the optimal policy function when X distance >= 10 considering the smoother policy at that location. The corners of the policy are unobserved states, which makes sense considering those represent where the pipes could have always been located. It is interesting to note that the Q-Learning agent has a better understanding for example it almost always flaps when on extremities below the pipe opening. However, neither agent learnt to stay ideal at the extremities above the pipe opening. This could possibly be because of the existence of situations where two nearby pipes have completely opposite pipe hole locations.

Fig. 3 in the Appendix shows that both agents beginning to learn to achieve high scores at around 6,000 episodes (Scores are used as a proxy to rewards as the plots were found to have the same trend). This is due to the epsilon decay set in both algorithms which motivate the agent to explore earlier episodes and once epsilon decay sufficiently, it begins to exploit greedy actions. However, the Expected SARSA shows more consistency in its learning of the optimal policy by having a significantly smaller score moving average variance. Hence, the second agent could be thought of as the more reliable agent to run over several game trials. Finally, it can be observed in **Fig. 4** in the Appendix that Q-Learning is more sensitive than Expected SARSA to hyperparameter selection given significant reward increase can begin in between 4,000 – 8,000 episodes whereas the latter has this upper bound to around 6,000 episodes. Of course, there are some pairs in both agents (generally with step size = 0.01) where the agent does not learn.

4 AGENT APPLICATION IN COMPLEX ENVIRONMENTS

4.1 Application in *TextFlappyBird-screen-v0*

The agent has been trained using an observation state space consisting of a tuple with dx and dy from the pipe gap (In this case pipe gap = 4 the second gap from the bottom). This environment outputs a state space as an array with the height and width of the screen. Whilst training the agent directly on the array of the space should be theoretically possible, it will likely be computationally very expensive. The easiest proxy would be to extract the distance from the pipe from the state space array. Another possible solution could be to make an approximation of the space in smaller state space. This would require a modification in the classes of the agents.

4.2 Application in Original Flappy Bird Setting

Running the agents in the original Flappy Bird setting could be problematic from a memory viewpoint. Kernels could crash immediately, and it would likely require very strong specs to run properly. Furthermore, the issue is that the observation is being outputted continuously (the observation of distance from the pipe are floats) rather than a discrete state array position from the pipe. Hence, the agents would not be able to run as they are in this environment and would agents capable of dealing with continuous observations in real-time or through approximation need to be used instead.

References

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

Appendix

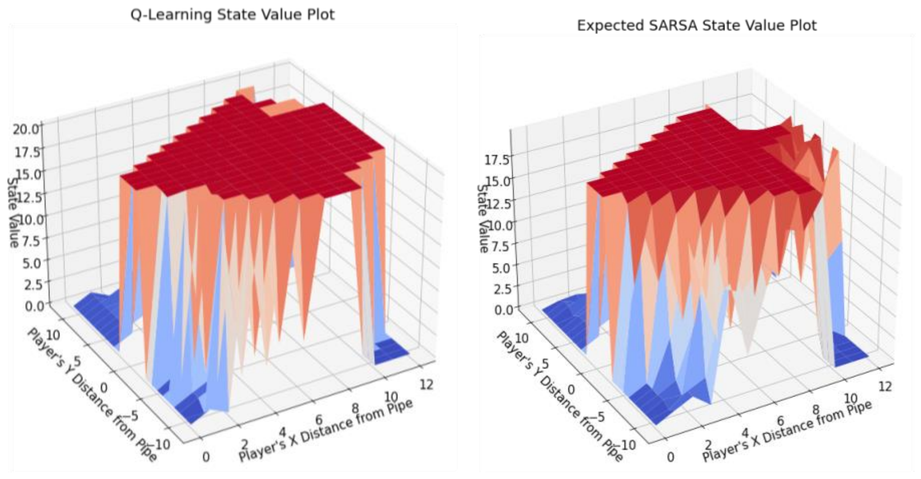


Fig. 1. Comparison between State Value Functions

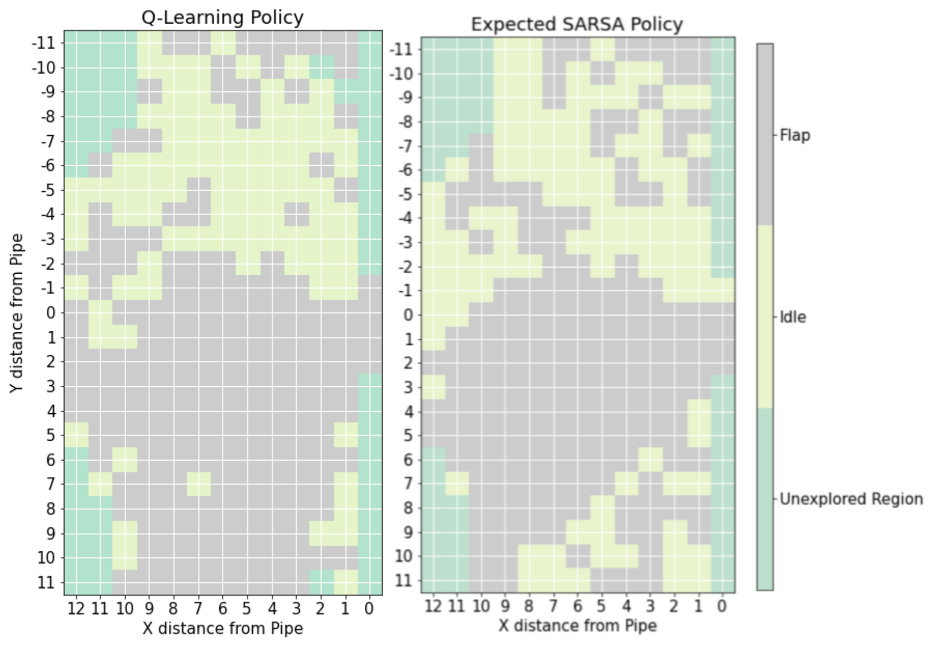
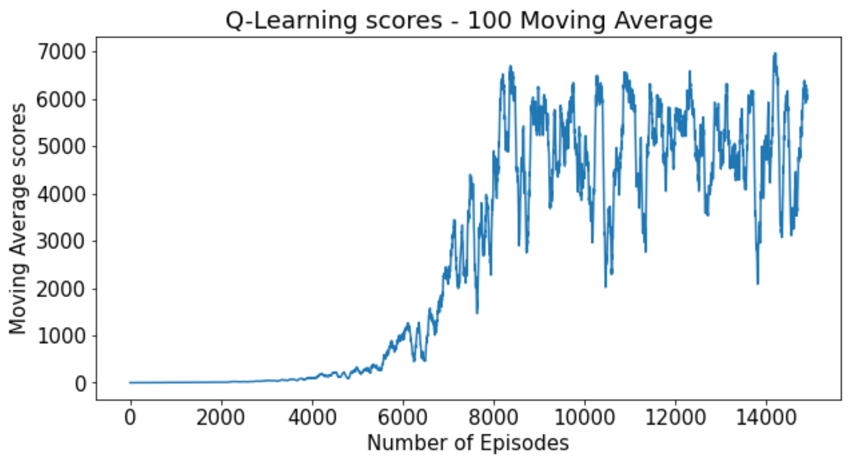


Fig. 2. Comparison of Agent Policies



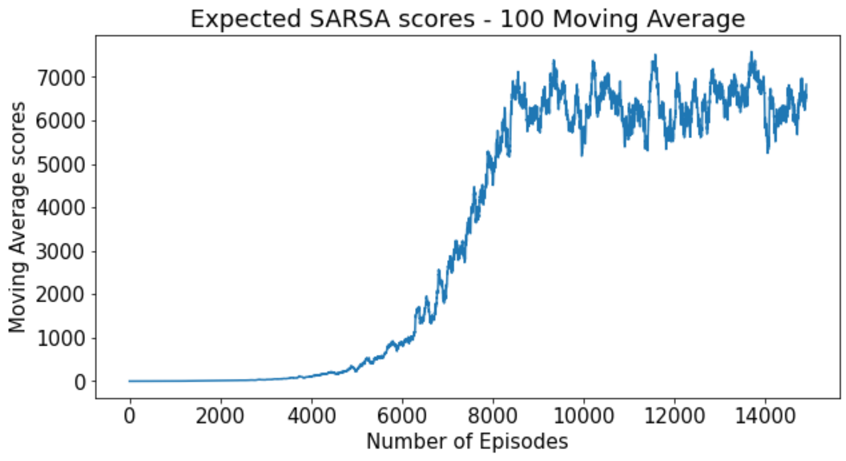


Fig. 3. Comparison of 100-Score Moving Average Per Episode

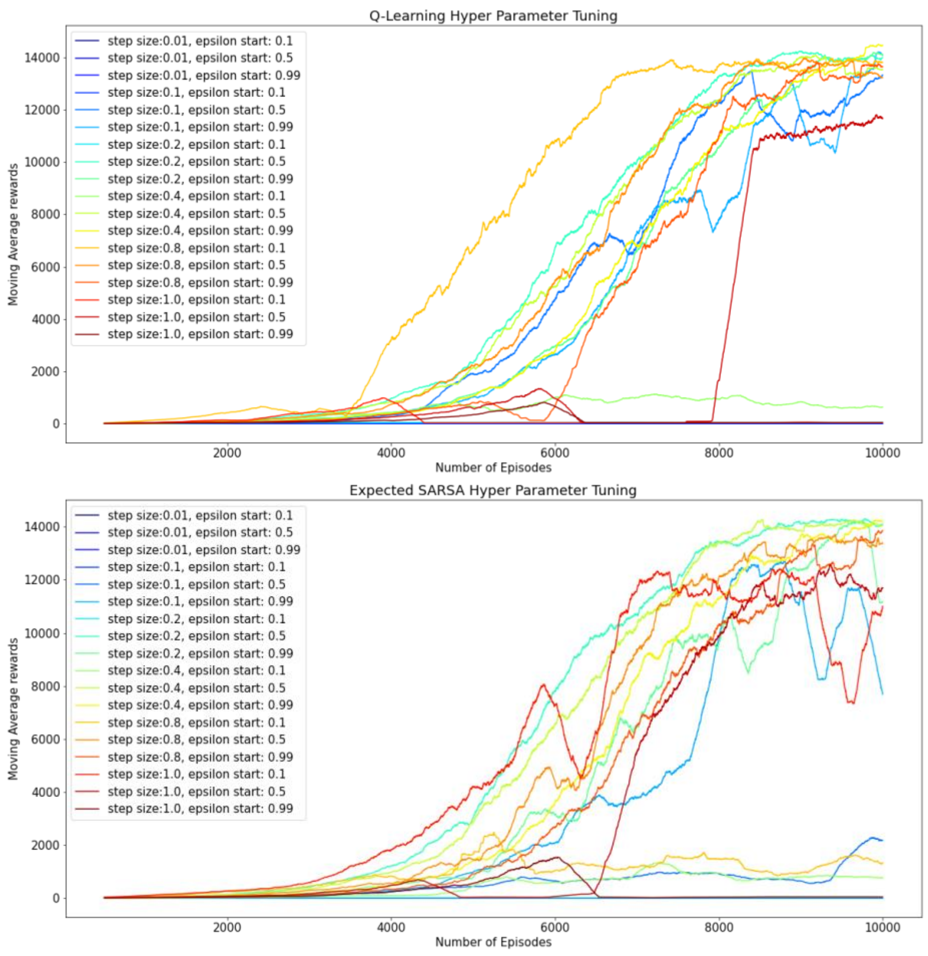


Fig. 4. Comparison of 100-Reward Moving as a function of different pairs of Step-Size and Epsilon Start