Flappy Bird Individual Assignment

Saureyj Verma

March 2022

Contents

1 Introduction **1**

2 Selection of agents **2**
2.1 Agent 1: Q-Learning 2
2.2 Agent 2: SARSA 2

3 Training the agents **3**

4 Results & Discussion **3**
4.1 Differences between Text Flappy Bird and Flappy Bird RL Game 9

1 Introduction

Reinforcement Learning is the science of teaching agents to learn and develop inference about their environments using the concept of reinforcement from psychology. The idea is that entities behave positively and perform more efficient actions if they are rewarded for the actions they take, and change their behaviour if they are punished for the actions they take. In terms of computing, the agent is a computer that learns to solve a complex problem by being rewarded and punished constantly in order to teach the agent the mechanics of it's environment and adapt to acquire the solution of it's problem within the environment.

In order to test the effectiveness of reinforcement learning, we test two different algorithms, Q-learning and SARSA in order to train the Flappy Bird and observe how the agent performs and learns the game over time. The code for the agents and the training modules for Flappy Bird have been referred in the Git repository[1]. The repository contains the following files:

- `main.py`: Used for training the RL agents for learning Text Flappy Bird.
- `agents`: Contains the Q-learning and SARSA agents, with a trainer module that is used inside `main.py` for training the two agents.

[1] https://github.com/clutchkingasiimov/FlappyBirdRL

- `Datasets`: Contains the saved Q-table for the agent's idle and fly action, including csv files for various parameter sweeps.

Note: The agents were initially trained in a `.ipynb` notebook but due to speed constraints and the code not looping efficiently, the agents were trained in the terminal and their outputs were saved in the `Datasets` folder and the respective plots shown in the report were made in `RL Graphs.ipynb`

2 Selection of agents

For the Flappy Bird game, what we observe is that the game is set in a dynamic environment where the probability of flying up or flying down when seeing an opening between two pipes is unknown. In other words, we do not have the transition probability of the agent when it takes an action and transitions from a state S to the next state S'. In environments where we do not have the transition states, the Bellman equation for value iteration cannot be used as the value iteration method requires us to have the transition probabilities of different states in order to find the optimal value of a given state.

Off the bat, we notice that Flappy Bird is an environment like this which has no transition probabilities of the next incoming state, hence it makes the environment stochastic where the bird's action can make it go high or low, but the exact height it gains or loses is something we are unaware of. Hence, in such cases using model-free agents that do not need to have the transition values for the states are used since model-free agents also converge to the true value of the state, but instead of relying on state value, we will use the action values to converge to the value of a given state.

2.1 Agent 1: Q-Learning

The first agent we implemented is Q-learning. Being an off-policy algorithm, the agent tries to minimize the difference between the Q-value of the current state-action pair and the maximum Q-value of the action taken in the next state. Being a temporal difference learning algorithm, Q-learning looks one-step ahead and tries to minimize the difference between the true Q-value and the approximated Q-value.

2.2 Agent 2: SARSA

The second agent we implemented is SARSA. Being an on-policy algorithm, SARSA behaves differently than Q-learning in terms of the Q-value it obtains for the next action as instead of the maximum action, it takes the action for the next state in the sequence and then minimizes the difference between the predicted Q-value and the true Q-value over time.

3 Training the agents

For training the agents, we set up both the learning algorithms in the Flappy Bird environment. Before the training initiated, we first defined the state and action variables:

- *Actions*: The agent in the environment has two actions; Flap (1) and Idle (0), both of them discretized in an action space.
- *States*: The agent's state is the (x,y) coordinate position of the bird in the 15×20 grid.
- *Reward*: The reward function returns a number for training the agent to learn how to fly. We used a custom reward function where the agent is rewarded +1 for surviving each state and -10 for dying since the initial reward function was not aggressive enough to punish the bird in having a long range of flight time.

Having our three environment variables defined, we then look at both our training agents and define their initial parameters under which they were trained:

- *Step-size*: The step-size α was kept at 0.8 in order to allow the agents to have a long-range look ahead before stepping towards the optimal Q-value
- *Discount factor*: The discount factor γ was kept at 0.95 in order to give the agents a higher incentive to account in for the values from the recent coming states. Smaller discount factors make the agents die faster since then the goal of the bird becomes to exploit rewards that make it fly the next (x,y) coordinate, and not the (x,y) coordinate where the opening of the pipe is.
- *Exploration-Exploitation factor*: To give our agents the leverage to explore new solutions, we set ϵ=0.05 in order to give the agents a 5% chance to change their action and explore their environment in search for better rewards. For SARSA, we apply an ϵ-decay of 0.001 in order to make the agent behave near-optimal policy.

After having our environment explored and the parameters set, we initialize the training phase of our agents for a total of 5000 episodes. The total training time of both the agents took upto 1 hour, with Q-learning training faster than SARSA in both speed and score. In the next section, we shall share the results of our findings.

4 Results & Discussion

After both the agents were trained, we observe their performance by storing their Q-values and the rewards each agent generates over 5000 episodes. Figure 1 show the average reward of the episodes over time. As we observe, we notice

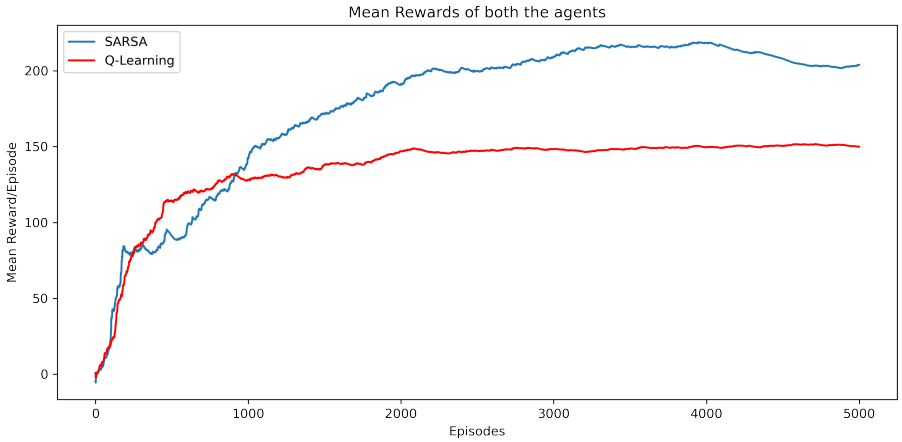

Figure 1: Average reward of both the agents over time

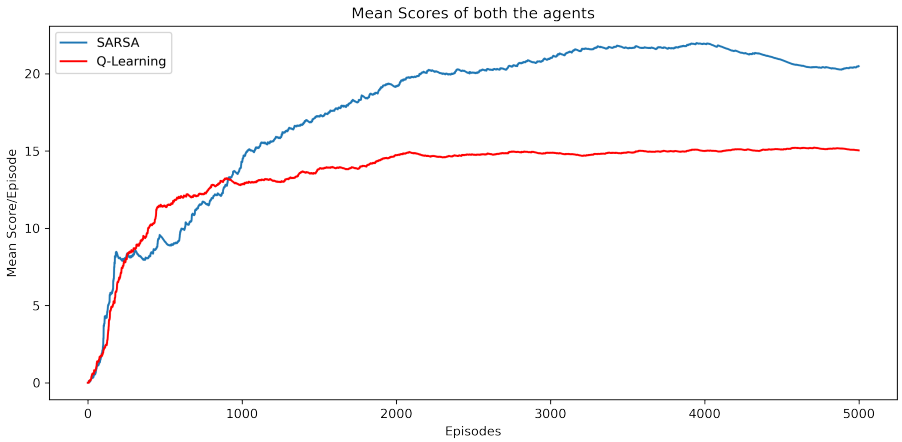

Figure 2: Average score of both the agents over time

that the average reward of SARSA is higher than that of Q-learning, with SARSA converging at an average reward of 200 per episode and Q-learning converging at 150 per episode. Noticeably in Figure 2, we see the scores of both the agents, which is also similar to the reward plot. Note that the score is the point given to the bird whenever it crosses an opening between two pipes, which means that a SARSA trained agent will, on average produce a flight with 20 points and Q-learning will produce a flight with 15 points. This does not mean that the bird cannot fly beyond 20 points and 15 points, but the graphs only show where the expected score of a given episode converges to. Further on, we also explore the distribution of the rewards and the scores and we find that for

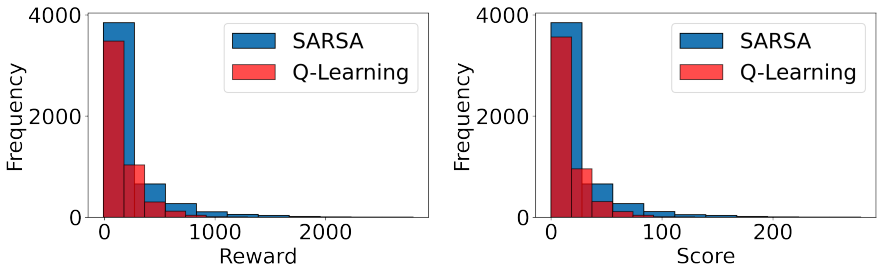

Figure 3: Distributions of the rewards and scores for both the agents.

the first 5000 episodes, the score and reward distributions are highly skewed, as observed in Figure 3.

Next, we observe the state-value plots of both the trained agents in order to see how their Q-values for a given state-action pair looks like. For each agent, we have 2 actions and multiple states, hence we create two heatmaps for each agent where one heatmap shows the Q-values for the Fly(1) action and the other shows the heatmap for the Idle(0) action. We observe the Q-value function for both SARSA and Q-learning below.

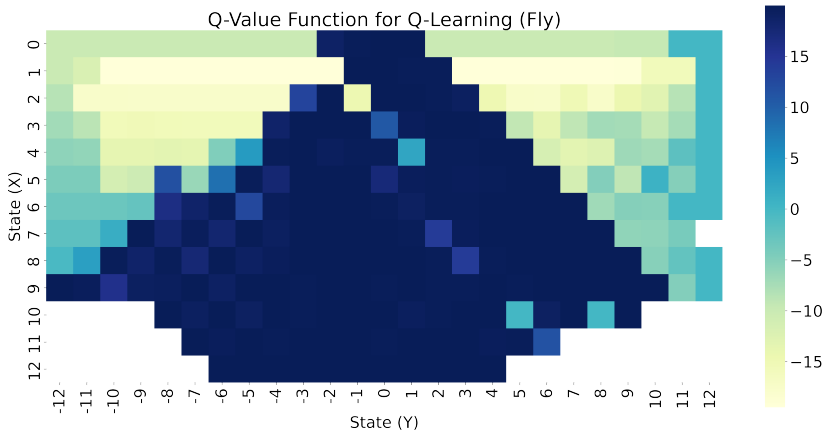

Figure 4: Q-values of Q-learning in Fly state (1)

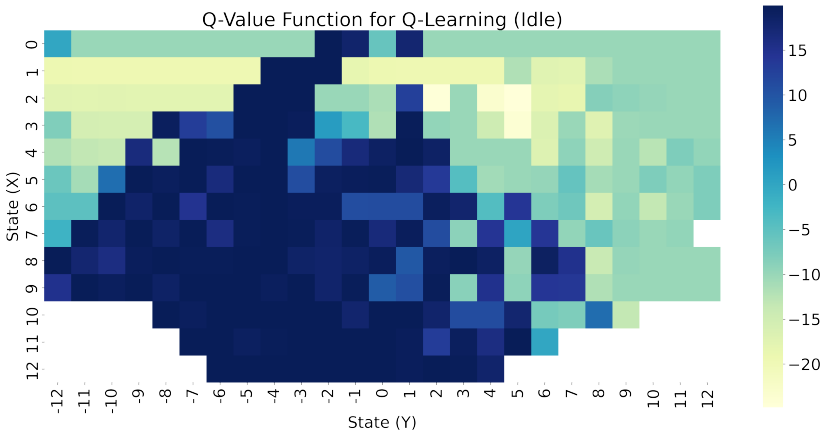

Figure 5: Q-values of Q-learning in Idle state (0)

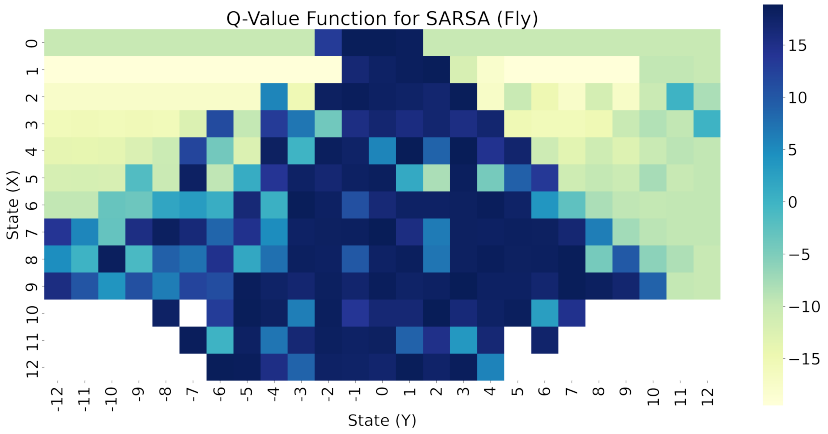

Figure 6: Q-values of SARSA in Fly state (1)

By looking at Figure 4 to Figure 7, we see the heatmaps and the four heatmaps reflect the inherent nature of the two algorithms and how the off-policy and on-policy estimation of Q-value happens during training. As we expect, Q-learning behaves in a much more greedy fashion compared to SARSA when we observe the Q-values for both Fly and Idle state as being off-policy, the Q-learning agent takes the maximum action from a given state and uses it to update it's next incoming action. In terms of Flappy Bird, the agent greedily selects the action that gives it the best chance to come close to the gap between the pipes, irrespective of how high or low the gap is. This means that even if the bird would need to fall down to a low position quickly or fly quickly to a high

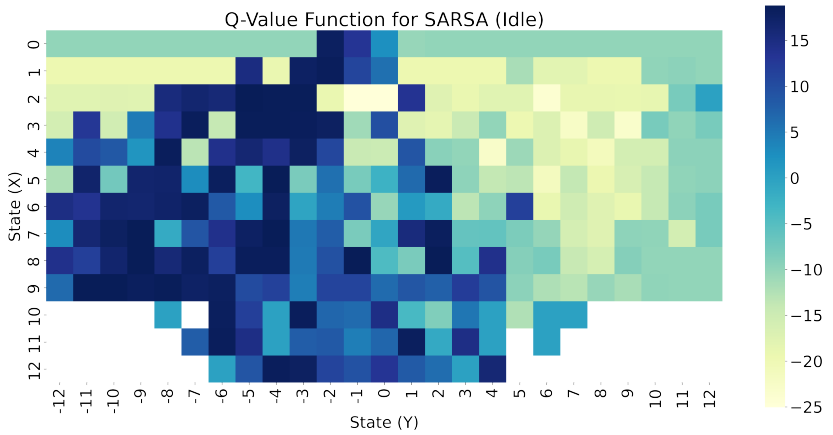

Figure 7: Q-values of SARSA in Idle state (0)

position, the bird will take that course of action simply because the off-policy approach makes the bird greedily choose the action that makes it maximize it's chances of obtaining the reward, hence it is by no surprise that the Q-learning agent goes for the aggressive manouvers at edge points of the 15×20 grid where the bird has a higher chance of dying.

Moving on to SARSA, the SARSA agent is far more cautious in it's learning phase, with the Q-values being updated less for coordinates in the grid where the bird has a higher chance of dying. The advantage of being on-policy here is that SARSA does not choose the maximum action for a given state, but rather sticks with the incoming action for the next state, hence the agent does not incentivize on always choosing the action that leads it to the gap between the pipes. It is interesting to notice that for SARSA, the Idle action state has Q-values that are lower than those of Q-learning. This itself displays the nature of on-policy algorithm and how the agent focuses on making survivability a long term target rather than score maximization using off-policy approach of greedy choice selection.

Next, we move on to parameter sweeps in order to see how a wide range of parameters impacts both the agents in their training phase. For parameter sweeps, we train our agents for 300 episodes only due the time it takes to train one agent effectively for 5000 episodes. For the three parameters, we tried the following range of parameter sweeps:

- *Discount factor*: We tried sweeps of values from 0.1 to 1.0, incremented in steps of 0.1
- *Step size*: We tried sweeps of values from 0.1 to 1.0, incremented in steps of 0.1
- *Exploration/Exploitation*: We tried sweeps of values from 0.01 to 0.1,

incremented in steps of 0.01

For both the agents, we performed parameter sweeps and totalled up the reward over the entire episode the agents receive.

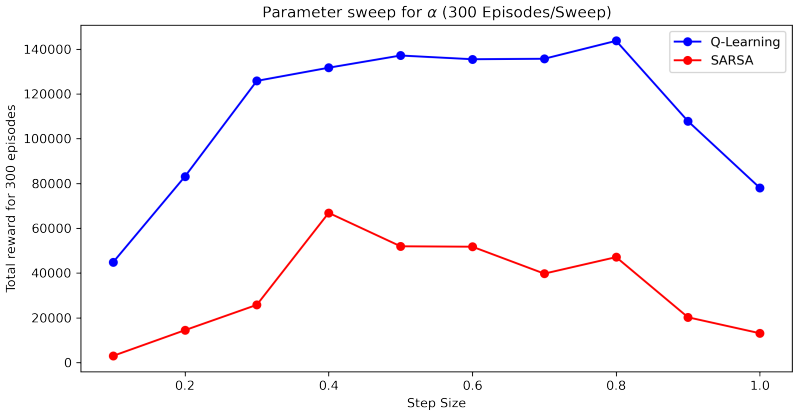

Figure 8: Parameter Sweep for α

For parameter sweeps done for step-size α, we observe that Q-learning performs the best compared to SARSA, with the best *alpha* at 0.8 for Q-learning, whereas SARSA performs the best at α 0f 0.4. In terms of step-size, Q-learning gives a higher reward with larger step sizes compared to SARSA since the agent chooses the maximum action each iteration, hence larger step sizes allow the agent to converge to the optimal policy faster. For discount factor γ, we notice

Figure 9: Parameter Sweep for γ

that larger discount factors lead to an increased reward amount for both the agents, which happens because larger discount factors allow the agent to focus on maximizing the Q-values for long term rewards that the bird will receive. For γ sweeps, both Q-learning and SARSA tend to perform better with higher discount factors.

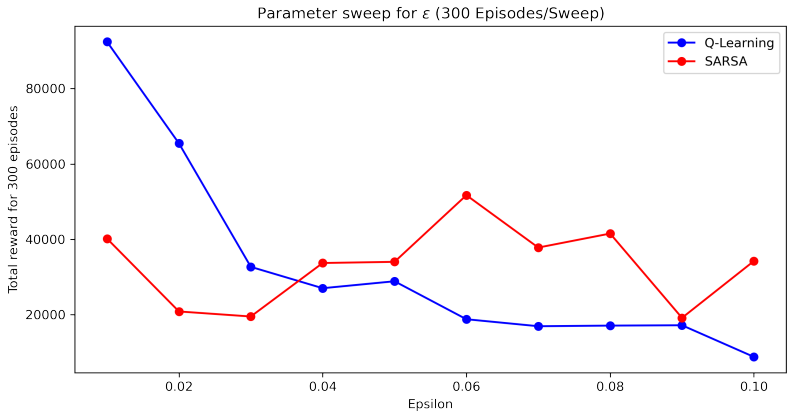

Figure 10: Parameter Sweep for ϵ

Lastly, we look at the parameter sweeps for ϵ to see how the rate of exploration and exploitation impacts the agents searching for the optimal policy. We observe that for Q-learning, an epsilon value of 0.01 leads it to having the best tradeoff between exploration and exploitation. For SARSA, we notice that an epsilon of 0.06 gives a good tradeoff between exploration and exploitation. We would also like to mention that for SARSA, epsilon-decay was implemented to make it perform on a near-optimal policy, henceforth the performance of SARSA is much better than that of Q-learning.

Due to time constraints, the parameter sweeps were kept for 300 episodes only but we believe that with a larger number of episodes, we expect to see a stronger asymptotic behaviour.

4.1 Differences between Text Flappy Bird and Flappy Bird RL Game

One of the reasons why the given environment was easy to work with is because the states and the actions are all discretized within numerical ranges, with the states being in the set of the state space of (x,y) grid, and the actions as Fly (1) and Idle (0). Compared to the Flappy Bird RL Game environment, the key differences that make both the game have different environments are:

- In the Flappy Bird RL game, the input is decomposed through a CNN in order to capture the state of the game, the states being the (x,y) position

of the bird relative to the gap between the pipe and the incoming distance of the bird from the pipe. This makes the problem more challenging as the state vector is now the position of the bird in a given frame of the game, but also provides us with a lot of information about the bird's flight.

- The estimation of the state-values will be much harder to do, but much more accurate and stable as instead of receiving a (x,y) coordinate itself, the agents will receive not only the coordinate position, but also the speed of the bird, the velocity of flying up or flying down, the distance of the bird relative to the gap between the pipes, etc. With a larger state space, the possible Q-values can have a higher dimensionality, for which sticking with regular model-free agents might not be the best option.

The environment of TFB has a much richer state space and much more information being provided due to each state representing the bird's current position in the frame. Comparing this with the text flappy-bird environment, the inherent limitation comes from the environment having a state space that only tells the position of the bird and excludes the various dynamics about the bird's flight such as it's speed, distance relative to the gap between the pipes and the ascent/descent rate. Because of the environment being dynamic and having multiple values in a given state, a better way to solve the Flappy Bird RL problem would be to use:

- DQN: With a DQN, we make the process of computing the Q-values much more tractable as now we can estimate the Q-value for many state-action pairs for an incoming state vector. For a game like flappy bird where flying or falling down can lead to many different flight positions in the game, DQNs can aid quite a lot in approximating the state-values.
- DDQN: We can perhaps also use a double DQN to speed up the training of flappy bird and use a game-cooperative approach to teach the agent the mechanics of the game.