3MD4120: Reinforcement Learning

Individual Assignment

Nhat Mai NGUYEN

nhatmai.nguyen@student-cs.fr

1 INTRODUCTION

Flappy Bird is a popular video game that gained widespread attention in 2014. The game's objective is to control the bird's flight by dodging the pipes to keep it alive as long as possible. It is an ideal scenario to build agents that can learn to play the game through reinforcement learning, a subset of machine learning that involves an agent learning to make decisions through trial and error. The agent learns by receiving feedback in the form of rewards or punishments for the actions it takes in a given environment. The ultimate goal is to find the optimal policy that maximizes the cumulative reward over time. In this project, I will use two reinforcement learning algorithms, Q-learning and SARSA, to train agents to play a text-based version of Flappy Bird. I will explore the difference between these algorithms and evaluate their performance in terms of reward & convergence time. This project serves as an opportunity to learn and gain hands-on experience in reinforcement learning.

2 METHODOLOGY

2.1 Environment

TextFlappyBird-v0 is a simplified version of the popular mobile game Flappy Bird. The game environment consists of a bird represented by the character '@' and pipes represented by '|' that move from right to left. The objective of the game is to navigate the bird through gaps in the pipes by taking either "Idle" or "Flap" action (corresponding to 0 and 1 in the notebook). The game ends when either the bird collides with a pipe or hits the ground (represented by '^'). The score increases by 1 for each gap the bird successfully passes through. The environment provides the state of the game, which includes dx: horizontal distance to the pipeline and dy: vertical distance to the pipeline gap center. At every step, if the bird is still alive, the reward returned by the environment is 1.

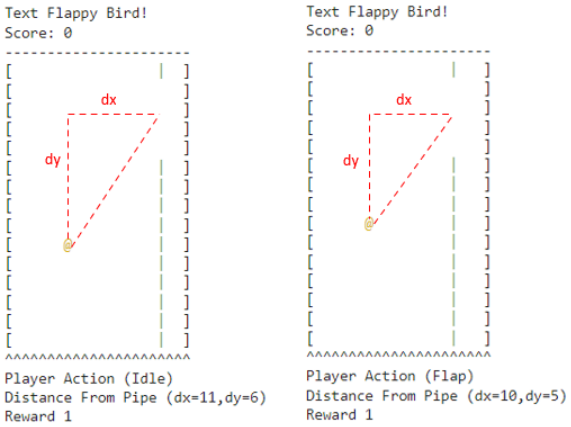

Fig. 1. A snapshot of TextFlappyBird-v0 game

2.2 Agents

Q-learning and SARSA are two widely used reinforcement learning algorithms for solving Markov decision processes. Q-learning is a model-free, off-policy algorithm that learns to estimate the optimal action-value function by iteratively updating the Q-values of state-action pairs using the Bellman equation. SARSA, on the other hand, is also a model-free algorithm that learns to estimate the state-action value function by iteratively updating the Q-values using the on-policy approach, where the next action is chosen according to the current policy. These algorithms have been used successfully in various game-playing scenarios, including classic games like Tic-Tac-Toe, Connect Four, and Chess, as well as more complex games like Atari games and the game of Go. They have also been applied to real-world problems such as robotics control, recommendation systems, and autonomous driving. The Q-value update formula for each algorithm is as followed:

Q-learning: $Q(s,a) \leftarrow (1 - \alpha)\ Q(s,a) + \alpha\ [r + \gamma \max(Q(s',a'))]$
SARSA: $Q(s,a) \leftarrow (1 - \alpha)\ Q(s,a) + \alpha\ [r + \gamma\ Q(s',a')]$

where $Q(s,a)$ is the estimated action-value function for state s and action a, α is the learning rate, r is the

immediate reward received after taking action a in state s, γ is the discount factor, s' is the next state, and a' is the next action.

Since TextFlappyBird-v0 has simple reward structure based on the number of pipes that the agent successfully navigates through, Q-learning and SARSA are a suitable choice for training an agent to play the game as both algorithms can learn the optimal policy for the agent through trial and error, by maximizing the expected reward in the environment. The reward scheme used to train the agents are + 1 if the bird survives and -10 if the bird dies. A random agent is also constructed for performance benchmarking purpose.

3 RESULT

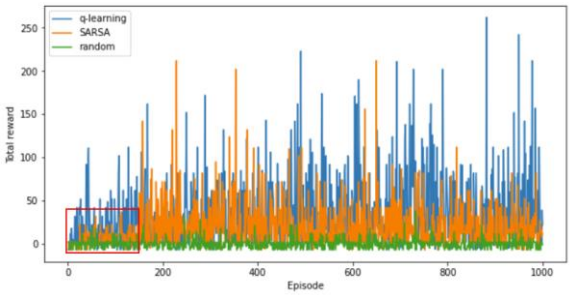

Fig. 2. Total reward in each episode by agent. Parameters used are α = 0.7, γ = 1, ε = 0.2, number of episodes = 1000

From Fig 2, Q-learning tends to achieve the highest total reward, followed by SARSA and then random actions. SARSA takes time to reach a good policy (see red box in Fig 2) and thus converges slower compared to Q-learning. While Q-learning considers the rewards of all possible long-term actions, SARSA considers the rewards of short-term actions based on current policy only which make it quite biased in the context of TextFlappyBird-v0 (a deterministic environment). As such, Q-learning performs better.

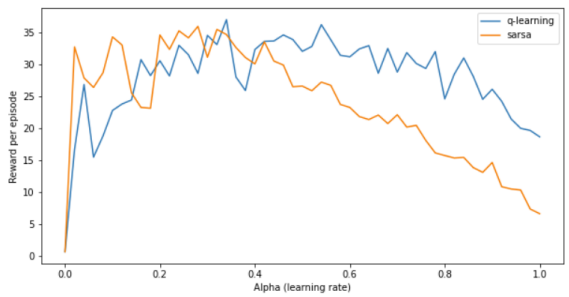

Fig. 3. Reward per episode by learning rate. Other parameters are fixed (γ = 1, ε = 0.2, number of episodes = 1,000, reward per episode = total reward / 1,000)

As expected, when the learning rate is too high, the agents overestimate the value of certain future actions and may fail to converge. Here, SARSA is more influenced by high learning rate because the considered future is short-term and biased towards current policy while in most cases, the bird supposes to consider several steps ahead to locate the pipe gap.

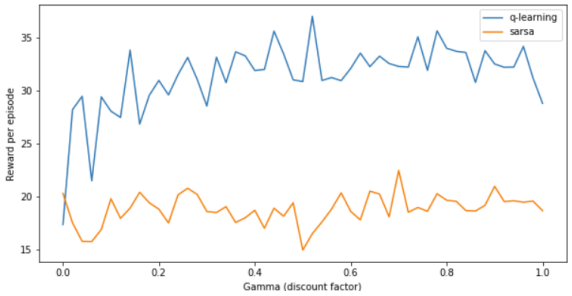

Fig. 4. Reward per episode by discount factor. Other parameters are fixed (α = 0.7, ε = 0.2, number of episodes = 1,000, reward per episode = total reward / 1,000)

In Fig 4, the trend is upward for Q-learning while it remains quite the same for SARSA. Q-learning considers long-term actions and high discount rate means the agent has a better overview of the environment (favors long-term rewards), thus performing better in term of reward but likely converging slower. On the contrary, SARSA only considers future reward from the next action. If the bird is far from the pipe, “Idle” or “Flap” mean the same (bird still survives, reward +1). In case the bird is next to the pipe, the bird has little chance to alter its action i.e. reward -10 (except that the bird is closed to the pipe gap, then “Idle” or “Flap” makes a different). As such, the reward of next action has little effect on the Q-values of SARSA. SARSA, therefore, is not sensitive to discount factor.

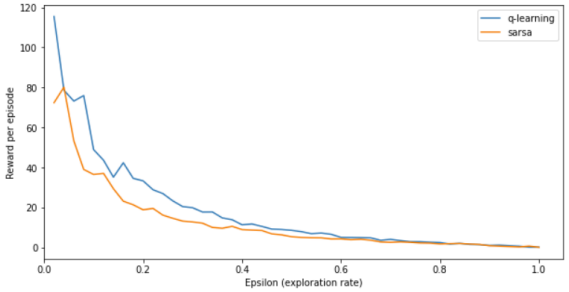

Fig. 5. Reward per episode by exploration rate. Other parameters are fixed (α = 0.7, γ = 1, number of episodes = 1,000, reward per episode = total reward / 1,000)

As the exploration rate increases, both agents intensively explore the environment, equivalent to taking random actions. The performance, thus, is low. At small exploration rate, the gap between the performance of Q-learning agent and SARSA agent can be attributed to SARSA agent likely gets stuck in the suboptimal policy. This is because SARSA agent selects actions based on current policy while Q-learning agent selects actions based on the maximum expected future reward.

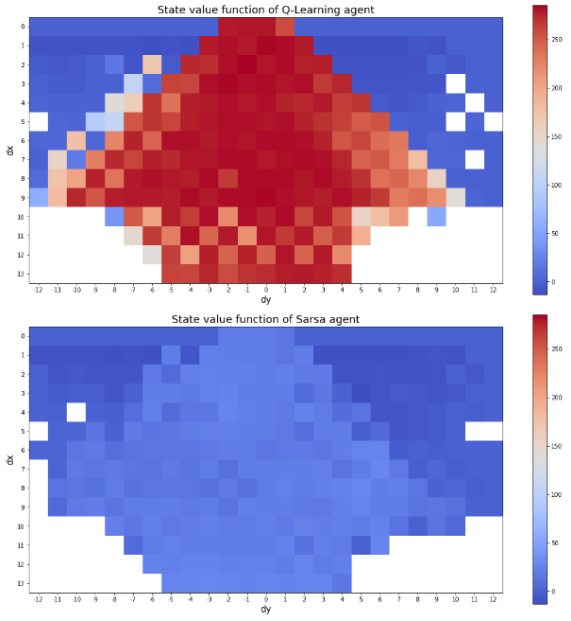

Fig. 6. State-value function by agent. Parameters used are α = 0.7, γ = 1, ε = 0.2, number of episodes = 1000

Aligned with previous observations, the values of Q-learning are higher than those of SARSA. The values are significantly high when the bird is in between the pipe gap (dy closed to 0). The high value area has an arrow sign, meaning that the closer the bird is to the pipe (dx closed to 0), the less chance it has to adjust its actions to navigate through the pipe gap.

4 OTHER ENVIRONMENTS

TextFlappyBird-screen-v0 is quite similar to TextFlappyBird-v0 except that the state returned by the environment is a matrix rather than dx, dy. In the matrix, the bird is represented by '1' while the pipes are represented by '2'. Given such matrix, we can construct a function to calculate dx & dy. Then, the above 2 agents Q-learning and SARSA can be implemented.

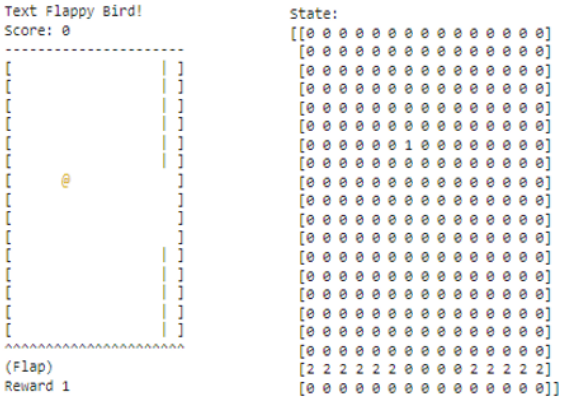

Fig. 7. A snapshot of TextFlappyBird-screen-v0 and its state

FlappyBird-v0 is an environment that simulates the popular mobile game Flappy Bird. It may look more complicated given that it has better user interface, audio, etc. but it is actually quite similar to TextFlappyBird-v0. The state returned also comprises the horizontal distance to the next pipe, difference between the player's y position and the next hole's y position (dx, dy). The actions and the reward scheme are also the same. As such, the constructed agents can be implemented in this case as well.

Fig. 8. A snapshot of FlappyBird-v0 and its states

RESOURCES

TextFlappyBird-screen-v0 and TextFlappyBird-v0

https://gitlab-research.centralesupelec.fr/stergios.christodoulidis/text-flappy-bird-gym

FlappyBird-v0

https://github.com/Talendar/flappy-bird-gym

The code

https://github.com/nguyen-nhat-mai/train_agent_play_flappy-bird