

Playing Flappy Bird with Two Different Value Learning Algorithms

Beining Li ^{1, †}, Yimeng Lu ^{2, †}, Yunhao Mo ^{3, †} and Weiqi Yu ^{4, †, *}

¹ School of Cyberspace Security, Sichuan University, Chengdu Sichuan, China

² Bell Honors School, Nanjing University of Posts and Telecommunications, Nanjing Jiangsu, China

³ School of Computer Science, China University of Geosciences, Wuhan Hubei, China

⁴ Bell Honors School, Nanjing University of Posts and Telecommunications, Nanjing Jiangsu, China

* Corresponding author email: yuwq@njupt.edu.cn

[†]These authors contributed equally.

Abstract. In this paper, reinforcement learning will be applied to the game flappy bird with two methods DQN and Q-learning. Then, we compare the performance through the visualization of data. Furthermore, more results from other games are summarized to analysis the corresponding advantages and disadvantages. Finally, we discuss and compare these two reinforcements learning methods.

Keywords: Reinforcement Learning; DQN; Q-learning.

1. Introduction

In reinforcement learning, we use a reward and punishment mechanism to train agents. When the agent makes the right choice, it is rewarded and the total score goes up, and when the agent makes the wrong choice, the total score goes down [1]. In this way, the agent will try to minimize its own wrong behaviour and maximize its own correct behaviour. In our life, reinforcement learning is used in many fields, especially in the field of engineering, such as driverless [2], finance [3,4] and games [5].

The reasons why we want to apply reinforcement learning in games are explained following. Playing games can be understood as making decisions about specific situations. Games are generally simulations of our lives and abstractions of scenes in life. Reinforcement learning of games is a simulation of reinforcement learning about specific situations in life. Compared with the real world, the game is simpler and less affected by factors. Only in the case of relatively few variables, the evaluation of different reinforcement learning algorithms can draw relatively more accurate conclusions.

The rules of Flappy Bird are as follows: In the game, the player controls a bird to cross obstacles composed of water pipes of different lengths. Players click on the screen control, each click on the touch screen, the bird will fly, keep clicking will continue to fly [6]. If you don't click on the screen, the bird will descend quickly. Birds will keep flying forward, and they need to pay attention to avoid obstacles on the way. One point is awarded for each post that the bird safely clears and misses. Of course, if you hit an obstacle, the game will end, and you will only have one chance.

In this article, we will introduce the implementation principle of DQN and Q-learning and the operation of flappy bird. The most important thing is to show our experimental results. We will use tables and images to make the comparison more intuitive.

2. Method

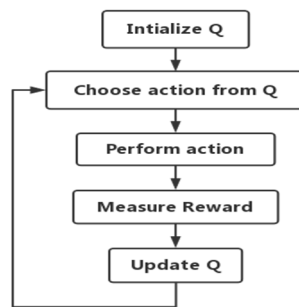


Fig 1. Algorithm flow chart of Q-learning

2.1 Q-learning

As shown in Fig. 1, Q-learning is one of the most basic algorithms for RL, whose feature is to record the Q value of an action in a table, where the actions performed by an agent in each state correspond to a value $Q(s, a)$ [7].

Q-learning algorithm committed to construct a Q-table to store Q-values which are generated from the states and actions of actors in the environment, and the agent selects the action that will yield the most profit based on the Q-values. The Q-table update process is shown in Figure 1.

We first initialize the value table ' $Q(s, a)$ ' and observe the current state ' s '. Next, an action ' a ' is chosen for that state based on a greedy strategy [8]. Each time an action is taken, a new state is created along with a reward, and the Q table [9] is updated by the following formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (1)$$

The next time the action is selected based on this updated Q-table. A near perfect policy can be obtained when all values in table are updated.

We apply q-learning to flappy bird. First, we consider that flappy bird has two actions: jump or not. We assume that action=1 means jump while action=0 stands for no jump. Each bird's distance from the next lower tube is called state. Our goal is to get more points for the flappy bird, that is, the longer the bird lives, the better. After each move, we set one point for survival and -1000 points for death.

2.2 DQN

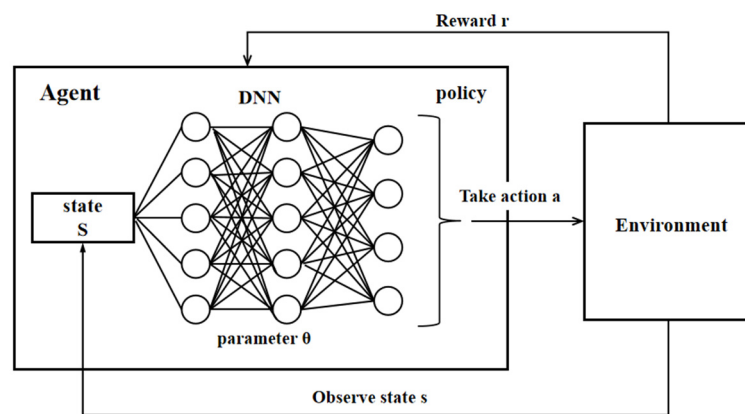


Fig 2. DQN neural network structure diagram

DQN refers to the Q-learning algorithm based on deep learning, which mainly combines Value Function Approximation and neural network technology, and adopts the method of target network and experience replay to train the network as shown in Fig. 2 [10].

When DQN is used on flappy bird, the state is a frame of images, and each state corresponds to two actions, which are all sent into the neural network. Q value is finally obtained through the analysis of neural network and the agent can choose its action according to the Q value. In addition, $(S_t, a_t, r_t, S_{t+1}, \text{terminal})$ is stored in the experience pool [11].

- S_t represents the state of the game at time t
- a_t represents actions performed by the game
- r_t represents a reward after an action has been performed
- S_{t+1} represents the new state obtained by executing actions in the state

Terminal is a sign that the game is over

The algorithm flow is as follows:

Table 1. Algorithm DQN

```

Initialize replay memory D to size N
Initialize action-value function Q with random weights
for episode = 1, M do
    Initialize state  $S_1$ 
    for t = 1, T do
        With probability  $\epsilon$  select random action  $a_t$ 
        otherwise select  $a_t = \max_a Q(s_t, a; \theta_i)$ 
        Execute action  $a_t$  in emulator and observe  $r_t$  and  $s_{t+1}$ 
        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in D
        Sample a minibatch of transitions  $(s_j, a_j, r_j, s_{j+1})$  from D

        Set  $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta_i) & \text{for non-terminal } s_{j+1} \end{cases}$ 

        perform a gradient step on  $[y_j - Q(s_j, a_j; \theta_i)]^2$  with respect to  $\theta$ 
    end for
end for

```

3. Experiment

In this experiment, to explore the effect and efficiency of optimizing algorithms applying on the Flappy Bird example, we chose two algorithms, Q-learning and its improvement, DQN, and adopted them to work on the Flappy Bird game sample. We have trained both algorithms and summarized by comparing the results.

In both Q-learning and DQN algorithms, in order to keep the consistency of the two algorithms, we set these following parameters at a same figure. Firstly, we set the decay rate, also known as gamma, at 0.99, in order to increase the weight of possible returns in the future. Then the alpha, also known as the learning rate, is set to 0.7. The epsilon of the epsilon-greedy method was set to 0.001, to make sure that there has a chance to let the agent explore the act that not have the maximum Q value, but not big enough to disturb the agent's action, because a random click could make the game fail even it is running stable before. Both agents interact with the environment once each frame, and the reward of bird keeping alive is set to 0.1 per frame, the reward of losing the game is set to -1. Respectively, for DQN algorithm, we set the replay memory at 50000 to obtain enough experience, decrease the loss of former experience and keep the stability.

We trained both the DQN and the Q-learning for 4000 episodes (an episode means a full round of playing game, from pressing the start button to the game over) and recorded the ability of the agent performing the game in different period of training. The following line graphs display the corresponding between the number of episodes training and the score of each algorithm the agent get.

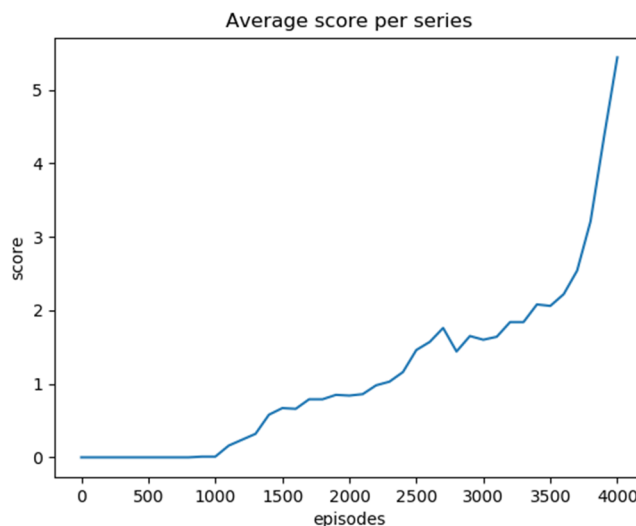


Fig 3. Average score per series (Q-learning)

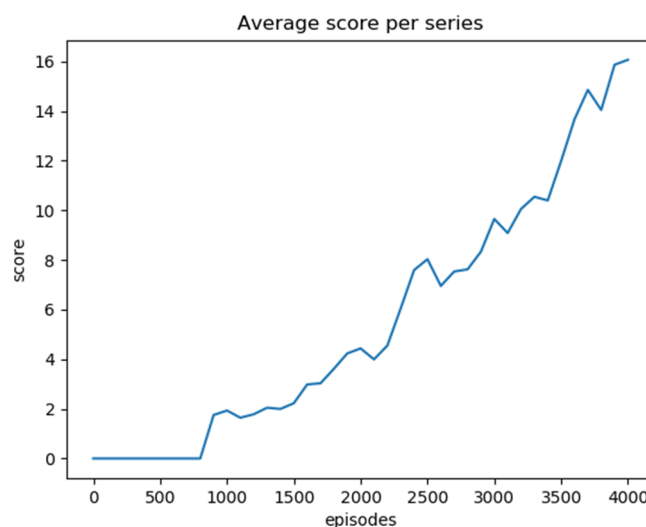


Fig 4. Average score per series (DQN)

In the two graphs, it can be seen that for the same figure of training episodes, DQN obtained almost three times as many scores to Q-learning and converged significantly faster than Q-learning. This means that the Q-network and the experience memory introduced by the DQN can effectively improve the efficiency of reinforcement learning on this example.

4. Conclusion

By studying, we can find that DQN is better than Q-learning in many cases. First, Q-learning and DQN are both off-policy algorithms. In the Q-learning algorithm, the agent needs to judge which action to take in a certain state according to the Q value, and the q table will constantly update its parameters according to the rewards obtained by the agent. But in high-dimensional complex problems, there are so many states that it is difficult to store them all in a table and searching in the table is also a time-consuming thing.

In DQN, the state and action are sent into the neural network, ensuing the Q value can be obtained after analysis. As a result, it is unnecessary to archive the Q value in the table. In addition, DQN also applies experience playback and fixed Q target. Every time we update DQN, we can randomly draw some previous experience for learning, which diverts high-correlated data into its low counterparts and equips the neural network with capability to update more efficiently. This is why DQN outperforms Q-learning in most games.

References

- [1] Zhang K, Yang Z, Başar T. Multi-agent reinforcement learning: A selective overview of theories and algorithms[J]. Handbook of Reinforcement Learning and Control, 2021: 321-384.
- [2] Shalev-Shwartz S, Shammah S, Shashua A. Safe, multi-agent, reinforcement learning for autonomous driving[J]. arXiv preprint arXiv:1610.03295, 2016.
- [3] Lee J W, Zhang B T. Stock trading system using reinforcement learning with cooperative agents[C]//Proceedings of the Nineteenth International Conference on Machine Learning. 2002: 451-458.
- [4] Lee J W, Park J, Jangmin O, et al. A multiagent approach to q -learning for daily stock trading[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2007, 37(6): 864-877.
- [5] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.
- [6] Liu B. Implementing Game Strategies Based on Reinforcement Learning[C]//2020 6th International Conference on Robotics and Artificial Intelligence. 2020: 53-56.
- [7] Kim K. Multi-Agent Deep Q Network to Enhance the Reinforcement Learning for Delayed Reward System[J]. Applied Sciences, 2022, 12(7): 3520.
- [8] Shaikh F. Simple Beginner's guide to Reinforcement Learning & its implementation[J]. Retrieved October, 2017, 11: 2018.
- [9] SADLAH S I A K. Studying Dynamics of Multi-Agent Learning in Networks[D]. The British University in Dubai (BUiD), 2012.
- [10] Sebastianelli A, Tipaldi M, Ullo S L, et al. A Deep Q-Learning based approach applied to the Snake game [C]// 2021 29th Mediterranean Conference on Control and Automation (MED). IEEE, 2021: 348-353.
- [11] Kim C, Park J S. Exploration with Multiple Random ϵ -Buffers in Off-Policy Deep Reinforcement Learning[J]. Symmetry, 2019, 11(11): 1352.