Using DQN and Double DQN to Play Flappy Bird

Kun Yang(✉)

School of Chemical and Material Engineering, Jiangnan University, Wuxi 214122, China
1051190130@stu.jiangnan.edu.cn

Abstract. Reinforcement Learning (RL), which is mainly used to solve sequential decision making problem, is an important branch in machine learning. Deep Learning (DL) also plays a leading role in the field of artificial intelligence. It uses neural network to approximate nonlinear functions. Deep Reinforcement Learning (DRL) is an algorithm framework combining reinforcement learning and deep learning that absorbs both advantages, and this DRL is capable of helping training agent learn how to play video games. Among them, the Deep Q Network (DQN) plays an important role. However, DQN will cause overestimation of values, and Double Deep Q Network (DoubleDQN) comes out and is used to fix this problem. The author presents the study of how DQN and DoubleDQN work and the difference between the result and training loss when these two algorithm were implemented in a video game called Flappy Bird in pycharm by using Keras (a neural network API in Python designed for deep learning problems). The author also adds improvement in DQN model to accelerate the training speed and compares it with the results of other people's experiments. After experiments, as expected, the modified DQN works better than traditional DQN but not as good as Double DQN. The training loss graph depicts that Double DQN decreases training loss present it is a good way to solve the overestimation problem.

Keywords: Deep Reinforcement Learning · Deep Q Learning · Double Q Learning · Artificial Intelligence

1 Introduction

Flappy bird is a video game which has some pipes preventing a bird from moving forward. Players need to control the bird to fly through pipes (there is a gap between the pipes) or the bird will die if it hits pipes. There are 2 actions that players can do: clicking the screen or doing nothing. If players click the screen the bird will fly higher a bit and if players do nothing, the bird will fall down a bit because of gravity. Players' goal is to control the bird to stay alive and move forward as far as they could. Every time players successfully fly through one pipe one score will be added on the scoreboard. This game can be played in http://flappybird.io and the game screen capture shows it in Fig. 1. Due to the fact that the agent can only do 2 actions and whether game over or not is easy to conclude, flappy bird could be a fundamental research about training artificial intelligence to play video game by deep reinforcement learning.

N. Radojević et al. (Eds.): ICAID 2022, AHIS 7, pp. 1166–1174, 2023.
https://doi.org/10.2991/978-94-6463-010-7_120

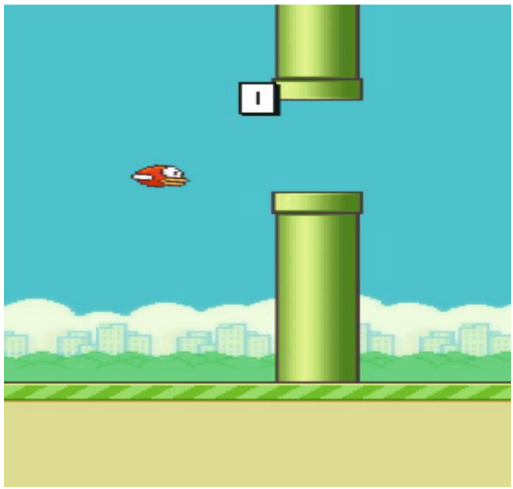

Fig. 1. Screen capture of flappy bird [1]

2 Related Work

Training an agent to play video games which is designed for human and comparing the level between agent and human is so fascinating that many people try to do this thing. VMnih et al. has shown that agents can be trained to play the Atari 2600 games by using deep reinforcement learning. DQN was used in their experiment [2, 3]. In DQN algorithm, a Deep Q Network is used to evaluate the Q-function in Q learning. However, using DQN will have an overestimation issue which may cause the convergence of training in a low efficiency. Hado van Hasselt et al. [4] illustrate the overestimation in the experiment training by DQN and modified DQN algorithm that turn it into Double Deep Q-Network to resolve this issue. In this project, the author tries to claim how DQN and DoubleDQN work, and uses these two algorithms to train agent to play flappy bird and compare the performance and training loss. Some improvements of DQN are also made.

3 Methodology

3.1 Reinforcement Learning

Reinforcement learning is a useful way for training an agent to act rationally in different environments, which is comprised by some states (s_t). The agent has to choose actions to interact with environment by observing the states and then it will get a reward or a punishment (r_t). Because of the actions did by agent, the environment might change, and the agent needs to choose actions again due to the new state of environment (s_{t+1}) and getting new reward or a punishment (r_{t+1}). This process shows in Fig. 2 [5].

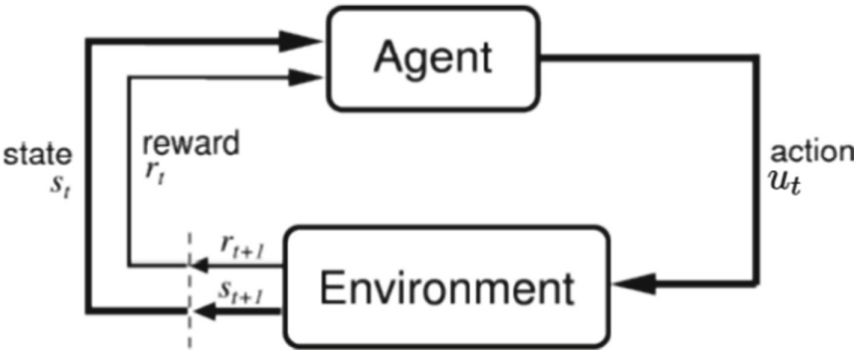

Fig. 2. Reinforcement learning process [5]

Many reinforcement learning situation can be described as Markov Decision Process that a quintuple including S, A, P, R, γ. S is a states sequence extract from environment. For example, the position of each Go chess piece can be the state when playing the game of go. A is actions that an agent can select to do. P is the probability of one state turning into another state. R is reward an agent can get when the transition is from one state to another (the value of reward is negative representing punishment). Since immediate reward is always important than future reward in real world situations, γ is set as discount factor to adjust the influence caused by future reward so the cumulative reward can be described rationally. γ equal to 1 means the author considers future reward as important as immediate reward. In Makiov Decision Process, the next state only depend on the previous state and the action, which is called Makrov property [5]. In order to compare which state is better easily, a value function is defined as:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{i=0}^{\infty}\gamma^{i} r_i | s_0 = s\right] \tag{1}$$

It means by using a given policy π, the expectation of the cumulative reward in state s as initial state s_0. In most instances, the author wants agents to select the action that can get the highest future cumulative reward so in the concepts of mathematics, the best policy π* can be defined as:

$$\pi^{*}(s_t) = argmax_{a \in A(s_t)} \sum_{s_{t+1}} P(s_{t+1}|s_t, a) V(s_{t+1}) \tag{2}$$

After agent does an action a the s_t will change to s_{t+1} in probability P ($s_{t+1}|s_t$, a), the best policy π* guides the agent to choose the best action according to maximizing the probability P ($s_{t+1}|s_t$, a) multiplying the next state value V(s_{t+1}).

Through mathematical manipulation the value function

$$V^{\pi}(s) = E_{\pi}\left[\sum_{i=0}^{\infty}\gamma^{i} r_i | s_0 = s\right] \tag{3}$$

can turn to

$$V^{\pi}(s) = \pi(s) \sum_{s' \in S} p(s, s') \left[r_0 + \gamma V^{\pi}(s') \right] \tag{4}$$

Which is called Bellman equation and the author gets

$$V(s) = r + \gamma \sum_{s'S} p(s, s') V(s') \tag{5}$$

when policy is stable. In fact, one policy can lead to many actions. In order to describing cumulative reward on account of one action a and one states, the Action-Value Function is defined as

$$Q(s, a) = r_s^a + \gamma \sum_{s \in S} p(s'|s) \sum_{a' \in A} \pi(a|s') Q(s', a') \tag{6}$$

from

$$V^{\pi}(s) = \pi(s) \sum_{s' \in S} p(s, s') \left[r_0 + \gamma V^{\pi}(s') \right] \tag{7}$$

In reinforcement learning issue, Q value of Action-Value Function can be an important reference of agent to select a optimal action in certain states.

In our flappy bird game experiment, S is composed by series of four consecutive screen capture as single state (since two consecutive screens capture show the bird's speed and direction, three tell us the bird's acceleration, four consecutive screen capture may be the best to describe the state in this game). [6] A is two actions the agent can choose. The bird will flap when a = 1 or do nothing when a = 0. If agent successfully goes through one pipe, reward is 1. However, if agent crushes the pipe or hits the ground and causes game over, it will get −1 reward as punishment. In order to encourage agent to stay alive longer, the author sets reward = 0.1 if agent survive in every time-step. γ is set equal to 0.99. The agent does not know P in this game. Our goal is to train the agent learn to select correct action by approximating the cumulative reward.

3.2 Q Learning

Q learning using a unique way to update Q function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma max Q(s', a') - Q(a, s) \right] \tag{8}$$

α is a coefficient called appropriately decreasing learning rate. Using bellman equation, if the reward is known, in finite states the author can draw a Q value table which depict convergence Q value in every action of every state and by choose the biggest Q value the best action is found.

3.3 Deep Q Network

The real world environment is complex, the possibility of state transition is unavailable, and the state may be infinite. Making a Q value table for a simple reality situation may be even impossible. Neural networks have proven to be a good tool to simulate nonlinear functions so that it can be used in Q learning to simulate Q function in infinite states. Combining neural network with Q-learning to realize deep Q-learning, a parameterized value function from a common loss can be learnt:

$$L_i(\theta) = E_{s,a\sim\rho(\cdot)}\left[\frac{1}{2}(y_t - Q(s, a; \theta_t))^2\right] \tag{9}$$

is used for function approximation, where define the parameters of the neural network in time-step t as θ_t and define the target in time-step t value as y_t. y_t can be represented as $y_t = E_{s'\sim\varepsilon}[r + \gamma \max Q(s', a', \theta_{t-1})|s, a]$

The gradient of the loss function with respect to the weights is

$$\nabla_{\theta_t} L_t(\theta_t) = E_{s,a\sim\rho(\cdot);s'\sim\varepsilon}\left[\left(r + \gamma max Q\left(s', a', \theta_{t-1}\right) - Q(s, a; \theta_t)\right)\nabla_{\theta_t} Q(s, a; \theta_t)\right] \tag{10}$$

So now the author could use stochastic gradient descent and back-propagation on the above loss function to update the weights of the network to approximate the Q function.

3.4 Double Deep Q Network

Though DQN performs well in the deep reinforcement learning which uses high dimensional as input to help agent make decision in large-scale problems, DQN has problems about overestimation of approximate function. In DQN, the agent chooses the action by

$$a_{opt} = argmax Q(s, a; \theta_t) \tag{11}$$

In addition, it finds target network y_t by

$$y_t = E_{s'\sim\varepsilon}[r + \gamma max Q\left(s', a', \theta_{t-1}\right)|s, a] \tag{12}$$

Time-step $t - 1$ is compare with time-step t when updating the neural network parameters, so actually the time-step that the agent chooses is the same as finding target network, which means that θ_t is equal to θ_{t-1} in the formula above. So the author actually chooses and evaluates actions in the same sets of weights θ. The author uses it to maximize the Q value twice. This leads the network resulting in overoptimistic value estimates, which may cause insufficiently flexible function approximation and noise [4]. In simple terms, it is dispensable for the best action of the next state which has the highest Q value [7]. In order to solve this issue, two parameters are used in DoubleDQN. One parameter θ is used to select action and the other parameter θ' is used to evaluate target network. The target network in DoubleDQN is:

$$Y_t^{DoubleQ} = r + \gamma Q\left(s_{t+1}, argmax Q(s_{t+1}, a; \theta_t); \theta_t^{'}\right) \tag{13}$$

After choosing the best action according to the main network with parameter θ_t, target network with parameter $\theta_t^{'}$ calculate the Q value due to this best action. Since Q value calculated by target network is not necessarily the highest one, Double DQN can decrease the probability of making sub-optimal overoptimistic choice.

Hado van Hasselt and Arthur Guez and David Silver make experiments to demonstrates that DoubleDQN indeed can successfully reduce the overoptimism of Q-learning [4].

4 Experiment and Discussion

The author implements DQN and Double DQN by in pycharm by using Keras.

4.1 Experience Replay

In video games, consecutive frames are much correlated. If neural network gets correlated information as input, the training of neural network will be inefficient. So experience replay should be used to de-correlate this situation [8]. Each (s, a, r, st + 1) as experience was stored in replay memory which has a certain size to only save the most recent experiences. The author uses a mini-batch comprised of certain size uniformly selecting experiences to train the neural network so the input correlation can successfully be reduced.

4.2 ε-greedy Approach

Due to the fact that if concentrating in exploration, it may wastes too much time to find a good action or if focusing on exploitation it may stuck into local optimum, there is an exploration-exploitation trades off which needs to handle. Thus the author chooses a random action with probability ε and chooses the optimal action $a_{opt} = \text{argmax}Q(s, a; \theta_t)$. Since the agent will be able to choose rational action after a large number of transtep, the ε will reduce from 0.1 to 0.0001 asymptotically in our experiment.

4.3 Model Architectures

The convolutional neural network used is learned from Mnih et al. [2], which is composed by three convolution layers and a maximum pooling layer and two fully connected layers. The input of the network is a 80 * 80 * 4 tensor from image captured from the game screen which is removed unnecessary background and converted to grayscale in order to train the network effectively. The input tensor is convolved with 32 filters of size 8 (stride 4) in the first convolution layer and then put into 2 * 2 max pooling layer. The second convolution layer is composed of 64 filters of size 4 (stride 2) and the final convolution layer has 64 filters of size 3 (stride 1). All these layers are separated by Rectifier Linear Units (ReLU). After that, the tensor is sent to fully connected layers successively, then the network exports the Q value of each action as its output. The author used Adma as the optimization methods to train the network.

Table 1. Time step and average score

	Modified DQN	Double DQN
Average score when 50000 training iterations	1.5	14.13
Average score when 100000 training iterations	84.6	102.17

Table 2. Time step and average score in [8]

Average score when 99000 training iterations	0.3
Average score when 199000 training iterations	11.6

4.4 Improvement

There are some reasons that may affect the training time so the author makes some additional improvement to accelerate the training speed. First, the author separately save the experience which reward equal to 1 in order to keep it when the replay memory is full. Because agent merely passes the pipe at the start of training, the successful experience to guide the agent is saved to choose the right action. Second, the agent chooses action every 5 frame rather than every frame. This implement has two reasons: one is choosing action in every 5 frame is more rational when comparing the result with human level because human can not choose action every frame with human reflexes; other is doing nothing in 4 frames in every 5 make agent easily to stay in the middle of the screen where the gap between the pipes appear in large probability so more successful experience in a faster speed can be obtained. Third, due to the previous experiment the author finds that agent always flies to the top of the screen and die. This may result of 0.1 reward that is set to encourage agent to stay alive. In the beginning of training, the agent will stay alive longer and get more reward if it always chooses fly action before it meets the first pipe. So the author sets more -0.5 reward if the agent touch the top of the screen as punishment to encourage the agent to get out of this dilemma.

5 Result

The experiment result implemented is much better than result in [8] shown in Table 1.

The overestimation problem in DQN is successfully solved when implementing Double DQN. Loss is defined as the difference value between the simulate Q network and Q target network. Two graphics are shown in Fig. 3 and Fig. 4 to intuitively understand the difference between implement Double DQN or not (Table 2).

As can be seen in Fig. 3, red line which represents the loss of training implemented Double DQN is manifestly lower than blue line which represent the loss of training implemented DQN. The loss data is logged to see the result more obviously which is shown in Fig. 4.

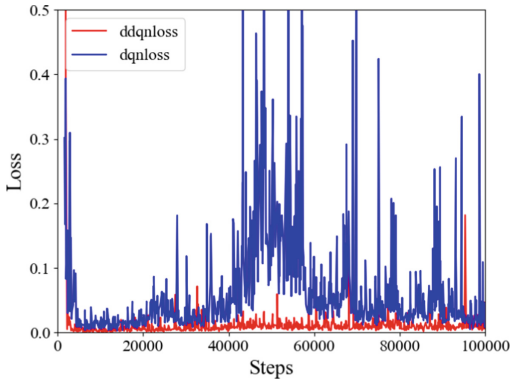

Fig. 3. Convergent loss in experiment

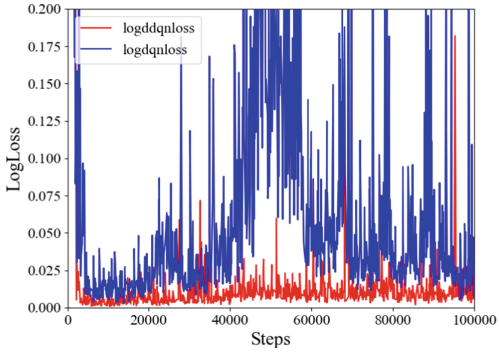

Fig. 4. Logarithmic loss in experiment

6 Conclusion

In this project the author illustrates the theory of DQN and Double DQN and successfully implement them to train an agent playing flappy bird. The improvement the author makes in traditional DQN accelerates training speed and the result of experiment shows Double DQN successfully settle overestimation problem in DQN. As for further study, the author separately saves the experience which successfully passes pipes as improvement to accelerate training speed but the fail experiences are also important to teach agent so it is interesting to find out that the appropriate number of successful experience should be set in experiment to minimize training time. Moreover, the author expects DQN and Double DQN could implement in more complex action choosing games rather than flappy bird whose agent has only two actions to choose. Overall, the author's experiment shows that the capacity and potential of deep reinforcement learning is implemented in video games.

References

1. 24 May 2018. http://flappybird.io/
2. Mnih V et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
3. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing Atari with deep reinforcement learning. In NIPS deep learning workshop
4. van Hasselt H, Guez A, Silver D (2015) Deep reinforcement learning with double q-learning. CoRR, abs/1509.06461
5. Rosset C, Cevallos C, Mukherjee I (2016) Cooperative multi-agent reinforcement learning for Flappy Bird
6. Pilcer L-S, Hoorelbeke A, D'andigne A (2018) Playing Flappy Bird with deep reinforcement learning. https://doi.org/10.13140/RG.2.2.13159.96165
7. Kong J, Shukla N (2018) Flappy Bird Hack using deep reinforcement learning with double Q-learning
8. Chen K (2015) Deep reinforcement learning for Flappy Bird