Reinforcement learning based AI in video game

Ruizhe Li, Chongyu He, Ning Ding
Virginia Tech
Blacksburg, VA 24060
`ruizhe@vt.edu, chongyuh@vt.edu, ningding@vt.edu`

Abstract

This project aims to learn and analyze reinforcement learning applications amount different games. We have picked three games that all use the Q learning algorithm. We have recreated the game environment and training for a similar result from the source for each game. The methodology used in the three games is being compared and analyzed during the recreation. Lastly, based on the finding, a well-constructed Q learning model that works for all three games was built.

1 Introduction

1.1 General Introduction

Reinforcement learning is a type of machine learning that the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment[1]. Video games are one of the most popular platform of testing machine learning like AlphaGo from DeepMind Technologies. By that inspiration, the team would like to test an algorithm that could runs multiple games with one model in Reinforcement learning. As a kind of of a reinforcement, Q-learning finds an optimal action in the sense of maximizing the expected value of the total reward over any and all successive steps, which has high value of application. Thus, Q-learning is our main method in our project.

1.2 Statement of the problem

This project aims to learn and analyze the application of reinforcement learning in games. The main algorithm chosen by this project is the Q learning algorithm. Another goal of this project is to generalize the model. To show our Q-learning model can be used generally. Our team members choose three different games. We use the same Q-learning code to train the agents in these various games. After training, The agent can finish the game at the end of the project. The final goal of this project is to build a Q-learning model that can be used for all three games.

2 Related Work

Q-learning In fact, Q-learning's research can be traced back very early. It was first proposed by Watkins in 1989.[2] In 1990s, many devices based on Q-learning has been developed.[3-7] In 2010, in order to solve the problem that Q-learning has large overestimation of action values.

Deep Q Network Deep Q network (DQN) combines Q-learning with neutral network. The important aspect of DQN is that it uses target network and experience replay. The target network is the same as online network, and the experience replay stores transitions for sometime, which can be used to update network. [8]

Double-DQN Double Q-learning was proposed by Hasselt.[9] Then, on the basis of his previous work. Hasselt combined existing DQN architecture and deep neural network of the DQN algorithm

.

without requiring additional networks or parameters, proposing a specific implementation called Double-DQN.[10] Double-DQN can have a better result in some games on Atari 2600 domain. Inspired by this this project uses Double-DQN to train our games' agents, hoping them to find best action policy to pass the game.

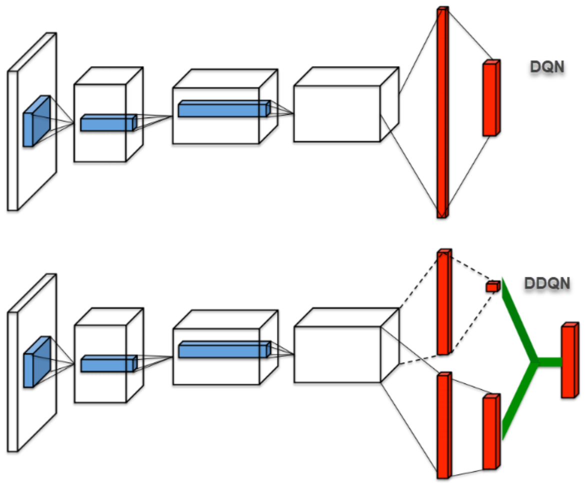

Figure 1: DQN vs DDQN [11]

3 Methodology

In this project, four elements are specific in our Double-DQN module: Environment, Action, State and Reward.Environment is what an agent interacts with and learns from.Action is how the agent responds to the Environment. State is the character the Environment shows. Reward is the feedback from Environment to Agent. Agent uses Reward to choose the future action.
Online network and target network are built in the model to get the right action after training.During training,two value functions are learned by experience, so that two sets of weights are updated. In each update one set of weight is used to decide the greedy policy and the other to determine its value.Just like what has been done in [10], the rule to update weight in our project is that greedy policy is evaluated according to the online network and the target network is used to estimate evaluation value.

The target value can be define as below:

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max Q(S_{t+1}, a; \theta_t); \theta_t^-) \tag{1}$$

Where, $Y_t^{DoubleDQN}$ is the target value,$\gamma \in [0,1]$ is a discount factor to show the importance of current and future rewards,R_{t+1} is current reward,S_{t+1} is future state resulted by current action,a is action,θ is weights of network, $Q(S_t, a; \theta_t)$ is parameterized value function.

Argmax is used to select the action and depends on the weights θ_t , which carry out the greedy policy. Second set of target net work weights of θ_t^- is used to evaluate the value of this policy.

4 Experiment

The gym library is a collection of test problems — environments — that we can use to work out our project's algorithm. [12] We choose games from gym, Acrobot System,Lunar Lander and Super Mario and build the project with "gym" package in Colabtory. According to the equation [1], we write the code to train θ_t , θ_t^- in order to maximize the target value and choose best action.Same architecture is used to different games.

4.1 Data and Environment

4.1.1 Acrobot System

The acrobot system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the target is to apply torques on the actuated joint to swing the free end of the linear chain(agent) above a given height(the black line in the picture).

4.1.2 Lunar Lander

Lunar lander is a game that user can control the lander which needs to land on the right position such as the center of the screen, by controlling the direction of the fuel. There are four actions: do nothing, fire left orientation engine, fire main engine and fire right orientation engine.

4.1.3 Super Mario

Super Mario is a game where the player controls a character called Mario, taking an adventure in the fictional mushroom kingdom. The character can run, jump, eat mushrooms, and step on the enemy's head from above. The character will die by falling off the platform, touching the enemy, or being hit by a fireball. After the character is dead, the player will have to start over again from the beginning. Finally, the player wins by going to the end of each map without deading. In this project, we use the 2D SuperMarioBros from the gym package.[13] It is a relative sample game with a complex moving environment.

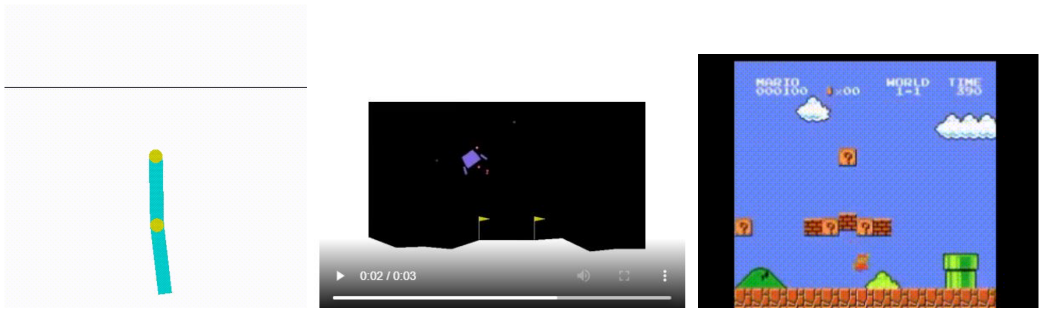

Figure 2: Acrobot System, Lunar lander, Super Mario

4.2 Training Process and Result

4.2.1 Acrobot System

The action is discrete, deterministic, and represents different torque applied on the agent's actuated joint. The action includes: (0) -1 Nm, (1) 0 Nm, (2) 1 Nm.The agent acts the best learned action at each of a sequence of discrete time steps, t=0,1,2,3,... . At each time step t, the environment give the feedback of the agent's state after agent's action. The shape of the state is 6, which indicates the position of the agent. The agent also receives a numerical reward from the environment. With the help of stage space and reward, Q-value is calculated at every step, helping the agent find a proper action to make itself in a new state. With the increase of the rewards, the agent will find the way to reach the target height by itself at the end of the game.

Acrobot System needs 800 episodes to reach convergence. We can see from the Figure 3 that with the increase of the number of episodes, the agent performers better and better to make itself go to a higher position. After 800 episode's training, it finally reaches its target line.

4.2.2 Lunar Lander

The landing platform is always at the coordinates (0,0). The coordinates are the first two numbers in the state vector. The reward for moving from the top of the screen to the landing pad and zero speed is about 100 to 140 points. If the lander moves away from the landing platform, it loses the reward. The episode ends if the lander crashes or stops, receiving -100 or +100 additional points. Each contact

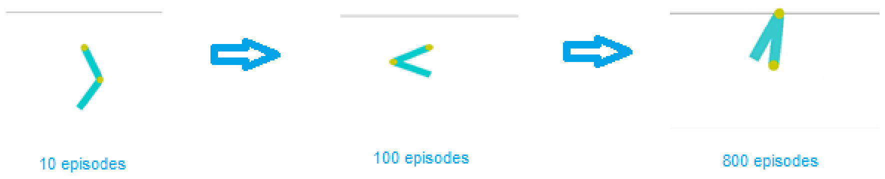

Figure 3: Acrobot System's Training Process

with the leg on the floor is +10. Firing the main engine is -0.3 points each frame. Resolved are 200 points. It is possible to land outside the landing platform. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Four discrete actions available: do nothing, fire the left-facing motor, fire the main engine, fire the right-facing motor.[12]

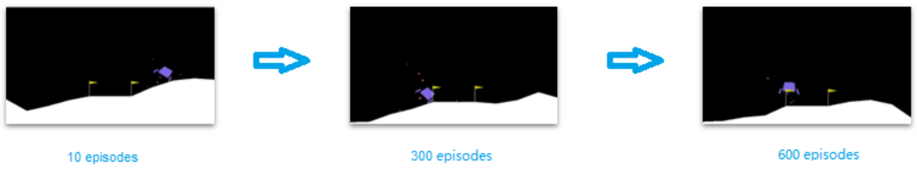

Figure 4: Lunar Lander's Training Process

After 600 episodes' training, Lunar Lander's convergence is satisfied. Figure 4 gives us the idea how Lunar lander chooses its action policy and lands on the landing platform finally.

4.2.3 Super Mario

Super Mario is the super star in game fields. In this game, Mario is the agent. The environment is the world that Mario is exploring. The action space is limited to move right and jump right. Reward is calculated through specific reward function provided by gym:

$$rewards = v + c + d \tag{2}$$

Where, v is the difference in agent values between states, c is the difference in the game clock between frames, d is a death penalty that penalizes the agent for dying in a state. The reward is clipped into range.

With the change of reward got at every training episode, Mario learns and updating the model until he wins the game.

Super Mario needs 50,000 eposides because of its complexity. During the training, Mario experiences

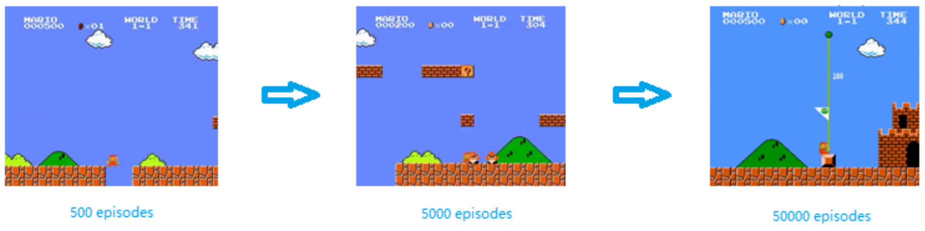

Figure 5: Super Mario's Training Process

different kinds of dying, such as jumping into a trap, touching enemies and so on. However, about 50,000 times' death is meaningful, which helps our model to update the θ_t and θ_t^- in equation [1]. In the end our model gives the best action in every state and Mario finally reaches the target, as shown in Figure 5.

5 Summary

Inspired by the Van Hasselt's research[10], we use Double-DNQ theory to train agents in three different games. Every agent has a good ending. We have three accomplishments from this project.

- We transfer the theoretical knowledge learned in the class into practical works. Through the work, we know how to analyze problems and write code according to theoretical equation;
- We become more familiar with machine learning research by doing literature review and also get inspired by others' research;
- We propose an architecture based on Deep-DNQ module, which can be used in three different games. We prove that our proposed architecture can train game's agent effectively, every of them can find proper action to obtain good result, no matter how complex the game is.

Contribution

Ruizhe Li: Contribute on lunar lander
Chongyu He: Contribute on Super Mario
Ning Ding: Contribute on Acrobot System

References

[1] Kaelbling, L. P., et al. (1996). "Reinforcement learning: A survey." Journal of artificial intelligence research 4: 237-285.

[2] C. J. C. H. Watkins. Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, England, 1989.

[3] Barto, A.G., Bradtke, S.J. Singh, S.E (1991). Real-time learning and control using asynchronous dynamic programming. (COINS technical report 91-57). Amherst: University of Massachusetts.

[4] Sutton. R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. Proceedings of the Seventh International Conference on Machine Learning. San Mateo, CA:Morgan Kaufmann.

[5] Chapman, D. Kaelbling, L.P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. Proceedings of the 1991 International Joint Conference on Artificial Intelligence (pp. 726-731).

[6] Mahadevan Connell (1991). Automatic programming of behavior-based robots using reinforcement learning. Proceedings of the 1991 National Conference on AI (pp. 768-773).

[7] Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8.

[8] Mnih, V., et al. (2015). "Human-level control through deep reinforcement learning." nature 518(7540): 529-533.

[9] Hasselt, H. (2010). "Double Q-learning." Advances in neural information processing systems 23.

[10] Van Hasselt, H., et al. (2016). Deep reinforcement learning with double q-learning. Proceedings of the AAAI conference on artificial intelligence.

[11] M. Lanham, Hands-on reinforcement learning for games: Implementing self-learning agents in games using artificial intelligence techniques. Packt Publishing, 2020.

[12] GYM Documentation, from:https://gym.openai.com/docs/

[13] Train a mario-playing RL agent. Train a Mario-playing RL Agent -PyTorch Tutorials 1.11.0 documentation. (n.d.). Retrieved March 16, 2022, from https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html