PLAY DAXIGUA USING DEEP REINFORCEMENT LEARNING

A PREPRINT

Luo Xinyang
School of Data Science
Chinese University of Hong Kong, Shenzhen
118020047@link.cuhk.edu.cn

May 21, 2021

ABSTRACT

We used deep reinforcement learning to train an agent to play Daxigua using diffirent algorithms. We tried value-based and policy-based method parameterized by deep neuron network. We added human expert trajectories and applied some techniques in reward function and algorithm to achieve better performance. Though the agents didn't exceed the performance of human expert, we found some potential problem and planned to improved them in the future.

1 Introduction

The target of this project is to apply deep reinforcement learning to vedio games. The game we choose is Synthetic Watermelon (Daxigua in the following context). In this game, each round a random fruit is generated, and the player is asked to drop it from a specific x location. When two fruits with the same type collide, the will become the one fruit of next type and recieve the corresponding reward. The ultimate goal is to maximize the total reward.
A common human approach is to place fruit with near type together and synthesis them all in one round, this need building a specific structure. Thus, to train an agent to play this game, we must carefully deal with credit assignment problem. The result might be used in other strategic game and scenario.

2 Related Work

One origin of playing vedio games using deep RL was the DeeepMind's paper in 2013 Playing Atari with Deep Reinforcement Learning Mnih et al. [2013] and their following paper in 2015 Mnih et al. [2015]. In these two paper, the authors used Convolutional Neuron Network(CNN) to extract feature from image input and proposed an algorithm called deep Q-learning with experience replay to train the agent. Besides, methods like Monte-Carlo Tree Search (MCTS) Guo et al. [2014] were also introduced to Atari game.
To deal with vedio games, few techniques were proposed to achieve a higher score. The design of rewards is essential, Zheng et al. [2018] introduced an intrinsic reward the help the agent to maximize the extrinsic (original) reward. O'Donoghue et al. [2016] combined policy gradient and q-learning. For some games that needs long steps to find the reward, Hosu and Rebedea [2016] added some human checkpoints that are close to reward for the agent to begin with. Some human trajectories are also added for the agent to learn fast. All the method proposed above achieve a better performance in soem Atari games.
However, unlike most of the Atari games which have fixed end states and determenistic transition mechanism, Daxigua has large randomness and no fixed end state, the game is like a spanning tree. Currently, there is no literature about playing Daxigua using RL, one similar game we can refer to is Tetris. Stevens and Pradhan [2016] used Q-learning and CNN to play tetris and achieved some results.

3 Methods

3.1 Implement Emulator

The original game is implemented using JavaScript, for interaction convenience we reimplemented it using pygame with pymunk as physical engine. This part referred to the code and API designed by Sharpiless [2021]. We used ball to represent the fruits, and for better image recognition, we use directly use different color to represent different type of balls.

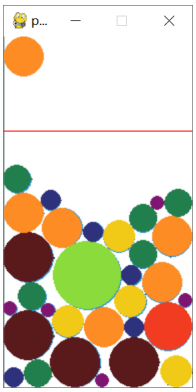


Figure 1: Sample of Emulator

Though the emulator is slightly different from the original game, it doesn't influence our study of applying deep reinforcement learning on this game.

3.2 Formalizing Markov Decision Process

We considered two ways of representing states. The first one is use raw image as states and apply CNN to extract features. The second one is manually feature engineering, first we initialize a $(p+2) \times (n+1)$ matrix with zero, where p is the number of type of balls, n is the number of features. Then we plug in the data. Each column consists of a one-hot factor of size p and location (x, y), which represents the information of a ball. The first column denotes the type of next ball and thus have location (0, 0). With this kind of matrix, we can use fully connected network in the later part. Although we have a lot of frames every secend, in the original game, we are only allowed to place the next ball after some timegap. Besides, the agent chooses its action based on the static position of each ball, there is no need to give them the states of the dynamics. Thus, agent gets state from the environment every three second and chooses actions. For action space, we used integer 0-187 to denote the actions, which correspond to the 188 pixel window width. Reward is the change of score after the action is taken.

3.3 DRL Algorithms

3.3.1 DQN

For this project, we first adopted method DQN in Mnih et al. [2015], the agent chooses its action by maximize Q-Value defined as

$$Q(s,a) = \mathbb{E}_{s'}[r + \max_{a'} Q(s',a')|s,a]$$

We use online network with parameter θ to estimate Q-Value of current step and target network with parameter θ^- to estimate the Q-Value of next step. We use replay buffer to store the transition, then we use batch training to minimize the loss with respect to θ

$$Loss = (Q(s,a|\theta) - r - \max_{a'} Q(s',a'|\theta^-))^2$$

The experience replay can recude the correlation among transitions.

3.3.2 Asynchronous DQN

To speed up our training process, we also referred to distributed DQN/ Asynchronous DQN in Nair et al. [2015] and Mnih et al. [2016]. Basically, we run our emulators and agents in a few processes. Then we accumulate gradients with respect to θ

$$d\theta = d\theta + \frac{\partial (y_i - Q_i(s, a, \theta))^2}{\partial \theta}$$

Every $I_{AsyncUpdate}$ step we update theta as distribute new parameters to the agent and every I_{target} step we update the paramters of target network using paramters of shared network.
Since our gradient is gathered from different agents, the correlation is low, thus we no longer need experience replay.

3.3.3 Actor-Critic

Since the action space is large, we tried to adopt some policy based method. Actor-Critic is what we chose. We use one network with parameter w to estimate value-function and another policy network with parameter θ. We first sample trajectory from policynetwork. Every step we do the following calculation and updating,

$$\begin{aligned} \delta &= r + \gamma \widehat{V(s', w)} - \widehat{V(s, w)} \\ dw &= \alpha_w \delta \nabla_w \widehat{V(s, w)} \\ d\theta &= \alpha_\theta \delta \nabla_\theta log_\pi(a|s, \theta) \end{aligned}$$

3.3.4 Network Architecture

For the image input, we used Resnet18 proposed in He et al. [2016] and for the matrix feture input, we used fully-connected(FC) network with 7 layers. We used Adam optimizer Kingma and Ba [2014] to update the parameters.

3.4 Modification to the Models

3.4.1 Human Designed Rewards

In the original game, reward will only be generated from the collision of two balls of same type. However, From human experience, it is important to pile up balls with adjacent type so that we can achieve high reward in later steps. Thus we designed an auxiliary reward, each time we put two balls of adjacent type, 0.5 reward will be generated (compare to the extrinsic reward range from 1 to 100).

3.4.2 Exploration

To encourage exploration, we first used epsilon greedy algorithm, starting from a high value(0.5 to 1) and gradually decay to 0.1. Also, we referred to the method used in DDPG Lillicrap et al. [2015], each step we add a random integer (from -2 to 2) to the action.

3.4.3 Imitation Learning

First, we pre-trained the DQN using 25 expert trajectories. While learning, we also add expert trajectories to the replay memory. The propotion of expert transition gradually decreases as number of training episodes increases.

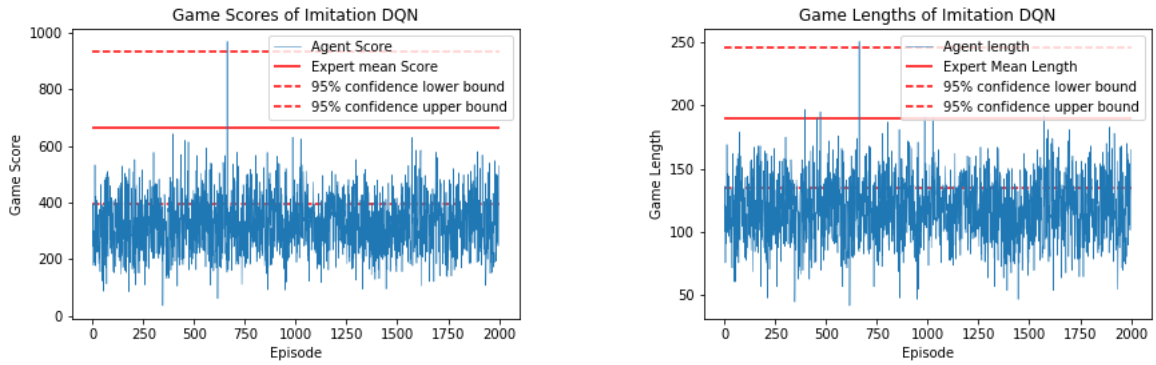
4 Training and Result

For Async-DQN we used matrix feature since it is trained on CPU, and for other agents, we used image input and trained them on CUDA. The Async-DQN is trained on 10 processes with 1000 episodes each. Others agents are trained for 2000 epsiodes.
Then we let each agents play 100 episodes with same random seed and observe their performances.

Agent	Average Score	Average Game Length
Async DQN	6.98	12.17
DQN with Auxiliary Reward	9.36	13.11
DQN without Auxiliary Reward	300.19	111.44
Actor Critic with Auxiliary Reward	6.98	12.17
Actor Critic without Auxiliary Reward	6.98	12.17
DQN with Expert Trajectories	373.88	132.52
Human Expert	663.04	190.16

Table 1: Performance of Different Algorithm

We also record training process of DQN with expert experience



(a) Game Score of Imitaiton DQN

(b) Game Length of Imitaiton DQN

Figure 2: Performance of Imitation DQN Compared to Human Expert

From the figures we can see that, the most of experience that the agent learns is from the pre training. In the later part of training, the curve doesn't show a upward trend.

5 Conclusion

We found that it is hard for agents to learn the game. None of them exceeds the averge score and length of human expert. Among all the agents, we found the imitation DQN has the best performance, which shows the importance of expert trajectories. If we use matrix feature, we found that the input is too sparse, and thus hard to found a suitable network architecture.

Through examing the trajectories of agents, we found that agents cannot assess the value of each action properly, most of the time they just repeat the same action ignoring what the state is. Besides, the game doesn't have an specific end state, and states are growing like a tree, which makes it harder for the agent to asses state and make action.It seemed that the agents doesn't summarize the game rule properly, which I thought is a consequence of limited training time and epsiodes. Although the dynamic of collision of balls is determenistic, the type of next ball is purely random (uniformly distributed from 1 to 5), the agent actions cannot influence this randomness which makes it harder to asses the Q-Value (in DQN) and Value-Function (in Actor-Critic).

6 Future Work

We attribute part of the poor performance to lack of exploration. The state space and action space is too large but each time we only have 2000 episodes. However, due to the limitation of CPU(the emulator must run on CPU), we can only train 2000 epsiodes in one day. In the future, we will try to increase the number of epsiodes to 10000 or even more. We also tried DQN with expert trajectories for 5000 episodes and still doesn't see any improvement, and unluckly lost all the data due to a windows restart.

We will try different algorithm like DDPG Lillicrap et al. [2015], A3C Mnih et al. [2016], PPO Schulman et al. [2017] and Monte-Carlo Tree Search. We also want to train a intrinsic reward network proposed in Zheng et al. [2018] to help the agent learn.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. *Advances in neural information processing systems*, 27: 3338–3346, 2014.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *arXiv preprint arXiv:1804.06459*, 2018.

Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

Ionel-Alexandru Hosu and Traian Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*, 2016.

Matt Stevens and Sabeek Pradhan. Playing tetris with deep reinforcement learning, 2016.

Sharpiless. Parl-dqn-daxigua. `https://github.com/Sharpiless/PARL-DQN-daxigua`, 2021.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.