Artificial Intelligence in Atari Freeway

Shang-Chi, Chen
109550110
chin2839211@gmail.com

Wei-Chieh, Tseng
109550156
weichieh.cs09@nycu.edu.tw

Hua-En, Lee
109550159
helee.cs09@nycu.edu.tw

Github repo link : https://github.com/Willy0921/2022NYCU-AI-Final-Project

Abstract

We trained an agent in an arcade game, Atari Freeway by different reinforcement learning algorithms to make it learn in an effort to perform as well as it could. Moreover, we gave comparison between the algorithms and analyzed the data to determine the cases in which our agent is best-performing.

1. Introduction

Atari-freeway is a game that a player can control a chicken to get to the other side of a ten-lane highway with heavy traffic flow. Once hit by a car, the chicken will be forced back slightly. While dodging the vehicles, we have to make it reach to the endpoint as fast as it can since our goal is to make the chickens that cross the road safely as many as possible within a limited time. That is, we must to train our chicken to march in an efficient way.

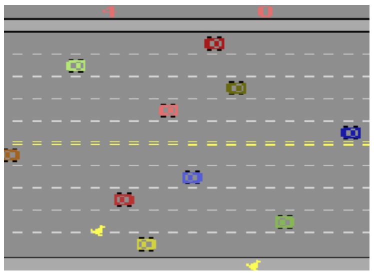

Figure 1. Atari Freeway

2. Related Work

- Count-Based Exploration with Neural Density Models, 2017 [1]
  Create an extra exploration bonus for a DQN agent and combine with Monte Carlo tree search.

- Deep Reinforcement Learning with Double Q-learning, 2016 [2]
  Apply the double Q-learning to reduce the overestimation of Q-value in Atari Games.

3. Methodology

First we installed Atari Freeway environment from OpenAI gym, which is an open source toolkit that has plenty of games environment for developers to train their AI.

Among the algorithms in reinforcement learning, we chose Q-learning and DQN to train our agent. According to the environment of Atari Freeway, we slightly modified the original algorithm.

3.1. Q-learning

When applying Q-learning, we first construct a Q-table of state and action which will be used for updating rewards.

Then we start to choose action according to the epsilon-greedy policy. We tend to make the chicken go ahead. Thus, we modified the epsilon-greedy policy to make the chicken have a higher chance to move forward.

After taking action and observing the results and rewards, we update the Q-value in our table according to the equation below.

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a)+\eta(r+\gamma\, max\hat{Q}_{opt}(s',a'))$$

However, in the environment of Atari Freeway, states are represented in 128 bytes. If we represent each state with 1 bit there will still be 2^{128} combinations. Since we have such a big Q-table and the agent only gets reward when reaching the endpoint, it turns out that we need a large number of episodes for learning to cover the whole Q-table. And 1-bit representation is too simplistic for the agent to learn about environment, but more bits means larger table.

In conclusion, the learning speed of Q-learning is too slow in Freeway.

3.2. DQN

For DQN, we use neural network to replace Q-table, which means that we can get features from the enormous state space directly. As we input an observation to our convolutional network, it will output a Q(s, a) for each action.

In the process of choosing action, we apply the same epsilon-greedy policy as Q-learning to urge the chicken go ahead.

By the algorithm mentioned above, we can restore a better experience in the replay buffer. Then we can have our agent sample from the buffer to learn. As for calculating the reward, once a chicken reach the endpoint it gets one point, which means the value of total rewards increases one. The addition of reward can be adjusted by setting the reward ratio, which is able to make it fit the Q-function sooner.

4. Experiments

By changing the parameters, we can get different results from our algorithm.

When initializing the neural network, Q value of three actions should be $(0, 0, 0)$, but due to unknown reasons, Q value would be three random number in the beginning. To solve this problem, we adjust the reward ratio to decline bias, so that the Q-function will be approximated well.

4.1. Results & Analysis

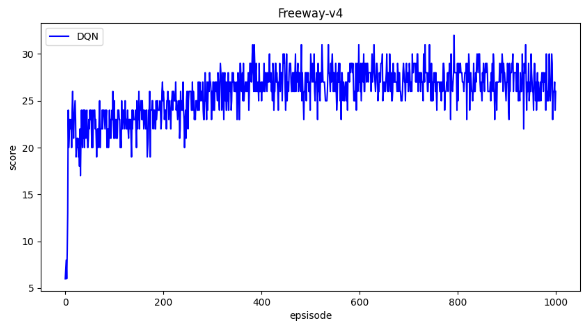

Figure 4. Agent's performance in DQN with episode = 1000

The graph shows that the score rapidly rise around the 20th episode, then it start to grow slowly. Around the 300th episode does the agent's performance converge to a certain degree that makes the average score about 29.

Since the average score converges in around 300th episode, we just have to focus on the first 300 episode when experimenting the difference of reward ratio.

$reward\ ratio$	Score
1	14.4
100	22.7
1000	**27.5**

Table 1. Average score of different reward ratio in DQN $(train\,time = 10)$

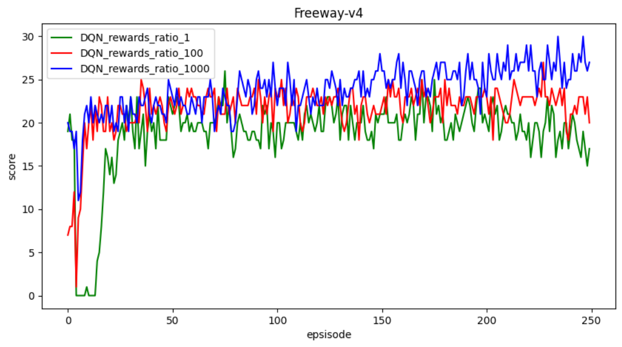

Figure 3. Agent's performance in DQN with different rewards ratio

Owing to the random bias causes by the initialized actions of Q-value, training with $reward\ ratio = 1$ can't make the agent take its best shot. Thus we try to train it with higher reward ratio, attempting to explore a better learning curve.

It can be observed that the score escalate dramatically in the early phase at each value of reward ratio. More specifically, with greater reward ratio comes with more rapid increment in scores and higher average score in the later period.

References

[1] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.