Project-DQN on Atari Game

Uppsala University

Reinforcement Learning

Ying Peng, Jiayan Yang, Yuehua Qin, Chuxin Zhang

22 May 2021

Contents

1 Description of DQN

DQN is an algorithm where a Q learning framework with a neural network proceeds as a nonlinear approximation on a state-value function. Unlike original Q-learning that requires a great amount of time to create a Q-table and memory to save history, which thus has its limitations in a large-scale problem, this novel algorithm has 2 main advantages to overcome these problems, i.e., experience replay and target network.

With replay buffer, DQN stores the history for off-policy learning and extracts samples from the memory randomly. Because in reinforcement learning, the observed data are ordered. Ordered data is prone to overfitting current episodes. So DQN uses experience replay to solve this problem. Experience replay stores all necessary data (state transitions, rewards and actions) for performing Q learning and extracts sample data from the memory randomly to update. It reduces correlation between data.

In addition, DQN is optimized on a target network with fixed parameters updated periodically so that the Q-network is not changed frequently and makes training more stable.

2 Configurations Results

2.1 Hyperparameters

In our experiments, we mainly tune the following parameters for comparison.

- Memory Size: Maximum size of the replay memory.
- lr: Learning rate, used for optimizer.
- γ: Discount factor.
- Using Win Memory: Whether we use another replay buffer, which will explain on section 3.3.
- Decay: Epsilon decay rate
- Average Reward: Evaluate the final model 10 times and average the rewards.

2.2 Results

All the experiments are implemented in Google Colab with 2.5 million frames. The parameters are explained as follows.

Table 1: Configurations and Results of Experiments

Experiment	Memory Size	lr	γ	Using win Memory	Decay	Average reward
1	10000	0.0001	1	No	30000	-14.5
2	100000	0.0001	1	No	100000	-5.3
3	100000	0.0001	1	No	30000	-5
4	100000	0.0001	0.99	No	30000	4.1
5	100000	0.0002	0.99	No	30000	7.9
6	100000	0.0001	0.99	Yes	30000	-3.8
7	100000	0.0002	0.99	Yes	30000	5.9

3 Discussion

The curve in the resulting figures may not be a good description of the performance of the current model, because we take the average of the most recent 10 episodes as the score of the current model. So when the experiment is over, we re-evaluated the average value ten times with the saved model. This result will be more representative.

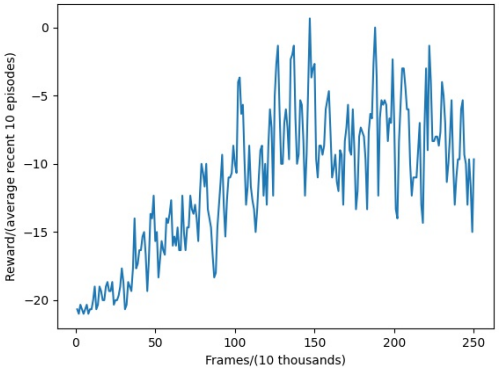
We implement multiple experiments based on the environment Pong-v0. In general, the results are basically satisfactory. The configuration of the model and its performance(Column *Average reward*) are displayed as Table 1.

3.1 Replay Memory Size

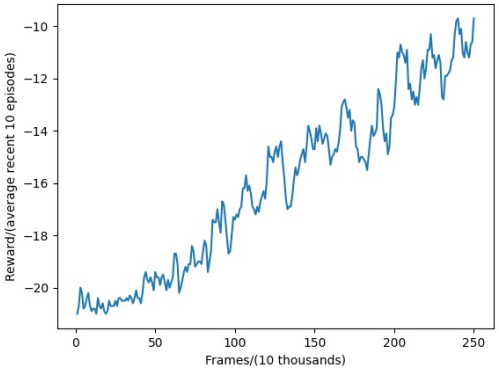
Figure 1 visualizes the results of Experiment 1, 2 and 3. It can be observed from 1a that when the replay memory size is 10000, the performance of the model is unstable, comparing with the averaged reward trend in Experiment 3. The reason for the differences is that the larger the experience replay, the less likely you will sample correlated elements, hence the more stable the training of the NN will be. However, a large experience replay requires a lot of memory so the training process is slower. Therefore, there is a trade-off between training stability (of the NN) and memory requirements. In these three experiments, the gamma valued 1, so the model is unbiased but with high variance, and also we have done the Experiment 2 twice, second time is basically satisfactory (as you can see in the graph), but first Experiment 2 were really poor which is almost same with Experiment 3. The result varies a lot among these two experiment due to the gamma equals to 1.

3.2 Learning Rate

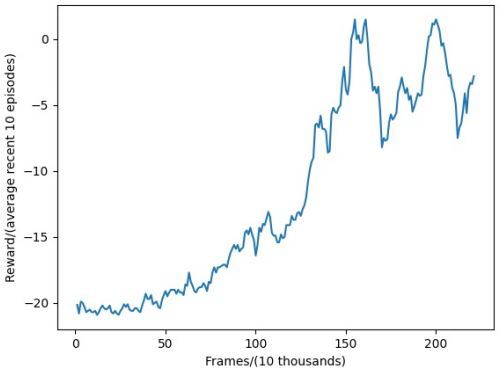
Now we discuss how learning rate affects the averaged reward. It is found from Figure 2 that a high learning rate has relatively large volatility on the overall curve, and the learning ability is not stable enough, but the learning ability will be stronger.



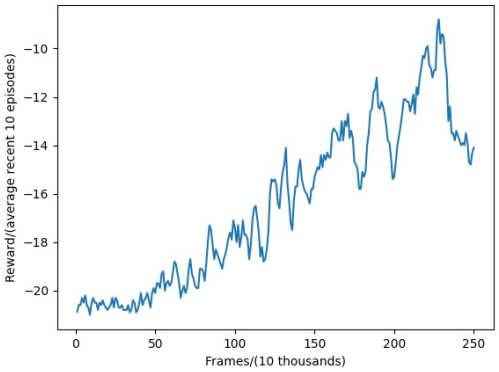
(a) Experiment 1



(b) Experiment 3

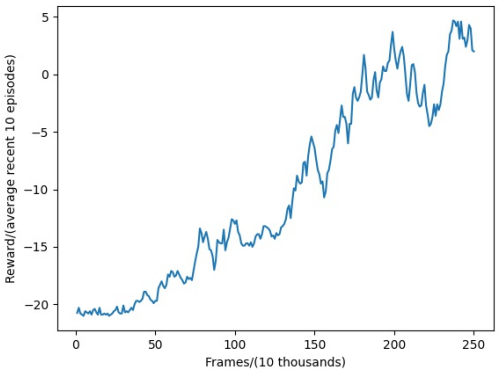


(c) Experiment 2(first time)

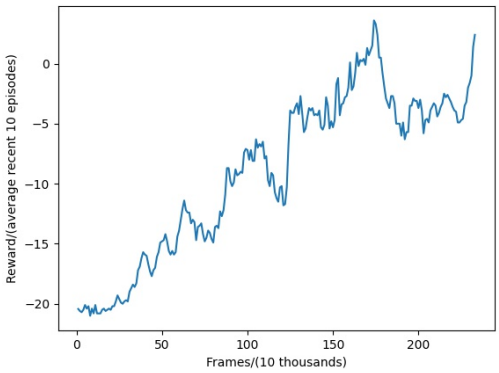


(d) Experiment 2(second time)

Figure 1: Mean Reward vs. Frames(Various Replay Memory Sizes)

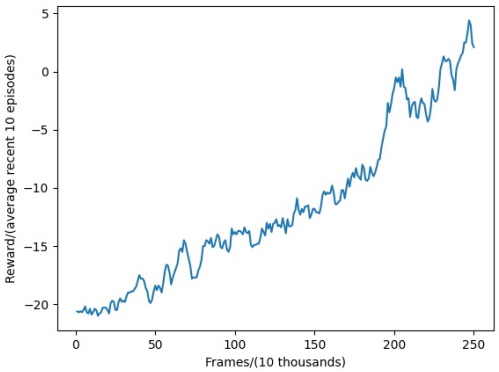


(a) Experiment 4

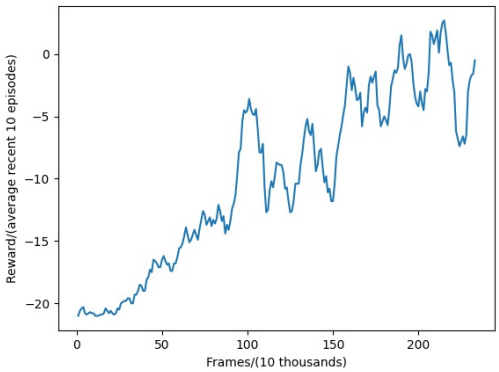


(b) Experiment 5

Figure 2: Mean Reward vs. Frames



(a) Experiment 6



(b) Experiment 7

Figure 3: Mean Reward vs. Frames(With Win Replay Memory)

3.3 Win Replay Memory

Here we try a new way to train our model and create a win replay memory for those frames that our agent gets reward 1. After 0.4 million frames, we start to randomly pick 5 samples from this win memory and then train the model every 5 thousand frames. The idea is for this kind of memory, the loss may vary a lot, so the model will tune the parameters more. But the results show that the performance is basically the same or even worse than that of learning rate = 0.0002.

3.4 Summary

Each experiment takes 4h on Google Colab. We achieve 10-time average reward of 7.9 as the best result that is better than Experiment 1(suggested configuration on Studium), although the result is somewhat random and may be unreproducible. It seems that the models with higher learning rate(0.002) perform better, but its reward influtuates more sharply.