Report

Lifeng Wei

October 2018

1 Introduction

This report summaries my explorations for solving the Banana Collector environment. First it briefly discuss the algorithms applied. Then it shows some of my experimental results for visualization and comparison between algorithms. In the last some possible improvements are also discussed.

The description for the environment can be found in the Readme.md file and will not be repeated here.

2 Algorithm

2.1 DQN

The full name of DQN is Deep Q-Learning. It is developed and applied by the DeepMind team in Atari games [2]. To understand DQN we must first understand original Q-Learning.

First we define some notations. For an episode we experience

$$\{S_0, A_0, R_1, ..., S_t, A_t, R_{t+1}.....\}$$

where S_t is the state at time t, A_t is the action we choose at time t and the reward we receive at time t is R_{t+i}. Besides, the function we use to estimate state-action values is $Q(s, a, w)$. w is the weights that will be learned from the whole process. At time t the weights would be w_t. Further more, we use γ to represent the decay rate for future rewards. And use α for learning rate.

The update rule for Q-Learning is:

$$w_{t+1} = w_t + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a, w_t) - Q(S_t, A_t, w_t))\nabla_w Q(S_t, A_t, w_t)$$

And for DQN, there are two main differences.

- Use a deep neural network to estimate the state-action values.

- Introduced the memory replay mechanism.

Next let's talk about them.

2.1.1 Network Structure

Of course different structures can be used. In this special case only fully connected layer and Relu activation [3] will be enough. The exactly structure I used is:

FC(statesize, 256)→Relu→FC(256, 128)→Relu→FC(128, 64)→Relu→FC(64, actionsize).

In this environment, statesize=37, actionsize=4. The network is defined as class 'QNetwork' in Net.py file.

2.1.2 Memory Replay

Memory replay means we store our past experiences and later we sample from them and do the update. To understand the importance of this mechanism, we need to first realize some import defects of Q-Learning:

- Low efficiency of using data.
- Consecutive samples are highly correlated
- Distribution of data is changing so very unstable

By memory replay mechanism the first two problems are alleviated. This is because each data can be sampled multiple times. And data points far from each other will have a smaller correlation. And to deal with the last problem, the following idea comes.

2.1.3 Two networks

In DQN there are two deep neural networks that share exactly the same structure. Let's call them *local* and *targer* network. Local network is updated by the update rule mentioned previously every time. And target network is updated toward local network with a smaller learning rate. By doing this the network which generates data changes much slower and the distribution also get stablized.

Next, let's talk about some improvement for DQN.

2.2 Double-DQN

Double-DQN[5] changes the function of local and target networks after several episodes: First we update local network and use target network to choose actions. After several episodes we update target network and use local network

to choose actions.

The idea of using Double Q learning is that Q learning uses the maximum value to estimate the future reward. But this could easily get over optimistic. Using one network to estimate the future reward while updating the other one can alleviate this problem because the two networks are not likely to over-estimate the same action.

This is not implemented

2.3 Dual DQN

Dual DQN [6] changes the structure of the deep neural network. Originally the network's outputs are estimates for state-action values. Now, from the second last layer, besides the existing fully connected layer, it adds another fully connected layer, whose output size is one. And it adds these two layers together to be the final output. See the figure below.

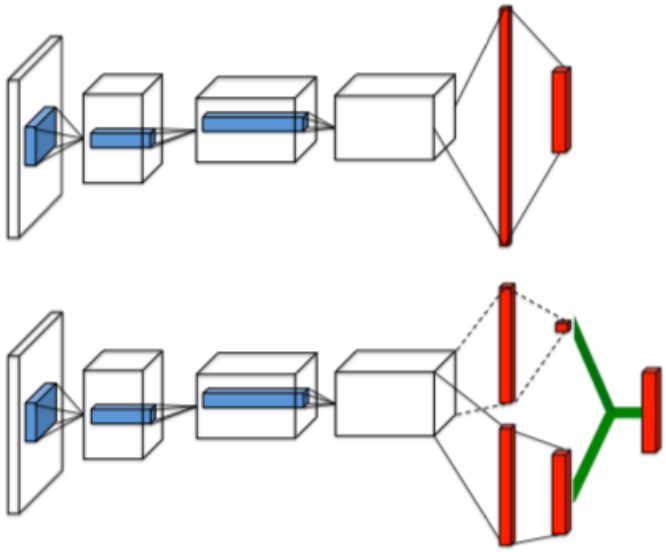

Figure 1: Difference between DQN and Dual DQN

This is implemented and the network structure is defined as class 'Dual_QNetwork' in Net.py file.

2.4 Prioritized DQN

Prioritized DQN[4] samples different experience with different probabilities. The idea is that if our original estimate is far away from the value we observed,

we should be able to learn more from that experience. So it assigns a larger probability for experiences with a larger difference. There are many other details in the original paper.

3 Experiments

In total I have tested four different settings. They are: two DQN with different learning rates, Dual DQN and Prioritized DQN.

3.1 Hyperparameters

Here we list all the hyperparameters' values. Their meanings will be explained right after.

Parameter	DQN1	DQN2	Dual DQN	Prioritized DQN
Learning Rate	1e-3	5e-4		
Buffer Size	1e5			
Batch Size	64			
γ	0.99			
τ	0.01			
Update gap	4			
Default Priority	NA			0.01

Table 1: Hyperparameters

Meanings:

- Learning Rate: α. The learning for local network
- Buffer Size: The number of experiences that re stored in replay memory
- Batch Size: The number of experiences 'replayed' when updating local network
- γ: The decay rate for future reward
- τ: The learning rate for target network
- Update gap: The number of actions taken between two consecutive updates for local network
- Default Priority: Only used in Prioritized DQN. Give new experiences the default priority.

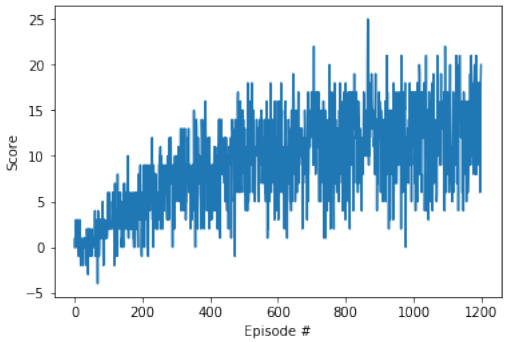

Figure 2: Score curve for DQN2. This curve solves the environment at episode 1107.

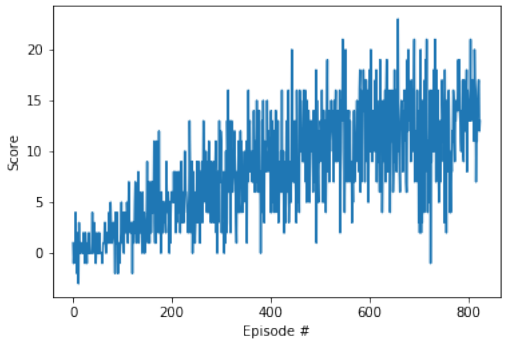

Figure 3: Score curve for Dual DQN. This curve solves the environment at episode 746.

3.2 Results

First we present one typical score curve for DQN2, Dual DQN and Prioritized DQN. The x-axis is the number of episodes and the y-axis is the score.

The one time result for reinforcement learning algorithms are unstable. To have a better comparison between different algorithms, we need a more stable quantitative method.

In order to find something more stable, I first calculate the mean reward for previous 100 episodes to get an 'average reward curve'. And for each algorithm I obtained 10 of this curve and take their average. This should provide us some idea of the expected value of average reward for different algorithms. See the graph below:

From the graph above we can see these things:

- Using same learning setting, prioritized DQN and Dual DQN has similar

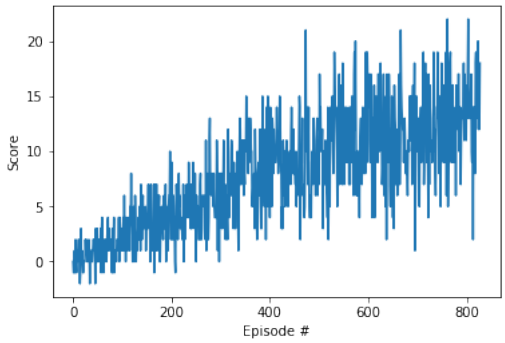

Figure 4: Score curve for Prioritized DQN. This curve solves the environment at episode 720.

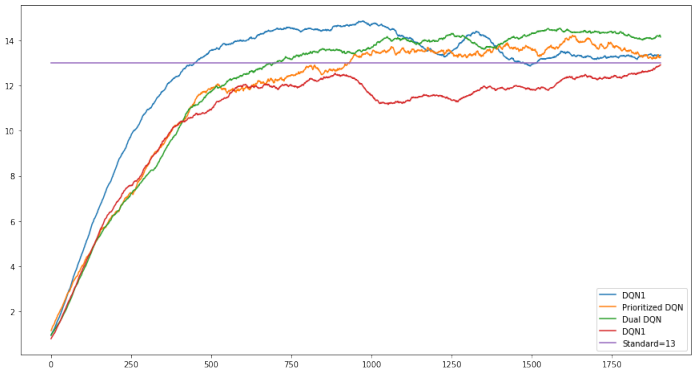

Figure 5: Averaged average reward curve for all four algorithms.

learning behavior and they are both much better and original DQN.

- Changing the learning rate only can make DQN learns much faster. But the drop in the end also shows the instable nature.

4 Future work

- Now the speed for prioritized DQN is much slower than the others. Will try to improve
- Implement Double DQN
- Implement Rainbow[1]
- Working on policy gradient now!

References

[1] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[3] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[5] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *ArXiv e-prints*, September 2015.

[6] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.