Report of Navigation Project

1 Problems

In a large, square world, an agent is wondering around and trying to collect bananas.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of your agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent has to learn how to best select actions. Four discrete actions are available, corresponding to:

0 - move forward

1 - move backward

2 - turn left

3 - turn right

The task is episodic, and in order to solve the environment, your agent must get an average score of +13 over 100 consecutive episodes.

2 Learning Algorithm

2.1 Deep Q-network

Deep Q-network (Mnih *et al.*, 2015), also known as DQN, is employed to solve this problem. Specifically, a neural network is used to approximate the optimal action-value function Q. The Q-learning update at each iteration uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s^{'}, a^{'}; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

To have a fixed target Q values in each update, two sets of parameters are stored θ_i and θ_i^-. Every 1000 steps, θ_i^- is updated with θ_i, and be frozen at other steps. The buffer size for experience replay is set to 10^5, and a batch of 64 transitions are sampled from the stored experiences and used for learning. The discount factor is chosen as 0.99 throughout the training. Adam is employed as the optimization algorithm to update the weights of neural network, with learning rate of 5×10^4.

Double DQN (Van Hasselt *et al.*, 2016) modifies the original DQN by using one Q network to select the current optimal action and the other Q network to evaluate the selected "optimal" action in each update.

Prioritized experience replay (Schaul *et al.*, 2015) further improve the learning process by sampling from the buffer experiences using the TD errors associated with each stored transition. And the sampling probabilities are calculated as:

$$P(i) = \frac{(|\delta_i| + \epsilon)^\alpha}{\sum_{j=1}^{N} (|\delta_k| + \epsilon)^\alpha}$$

where δ_i is the TD error and N is the total number of transitions that been stored currently. In this project, α is set to 0.6, and ϵ is set to 10^-6.

To compensate the bias due to un-uniform sampling if use original DQN update rule, a weight need to be applied to each sampled transition as,

$$w_i = (NP(i))^{-\beta}$$

In this project, β is set to 0.4 initially, and anneal to 1 each update.

2.2 neural Network Architecture

A three-layer fully connected neural network is employed to estimate the action-value function. Input size is 37 (dimension of each state in the environment), followed by two hidden layers with 128 and 64 neurons respectively, and finally gave a 4-dimensional vector, which represent for the estimated action-values for all possible actions under the given state.

3 Results

3.1 DQN

The plot of reward per episode is shown in Figure 1. In total, 2000 episodes are run.

3.2 Double DQN

The plot of reward per episode is shown in Figure 2. In total, 2000 episodes are run.

3.3 Prioritized Experience Replay

The plot of reward per episode is shown in Figure 3. In total, 1000 episodes are run.

3.4 Summary

- DQN need 1500 episodes to solve the problem.
- Double DQN need 900 episodes to solve the problem.
- DQN with prioritized experience replay need 800 episodes to solve the problem.

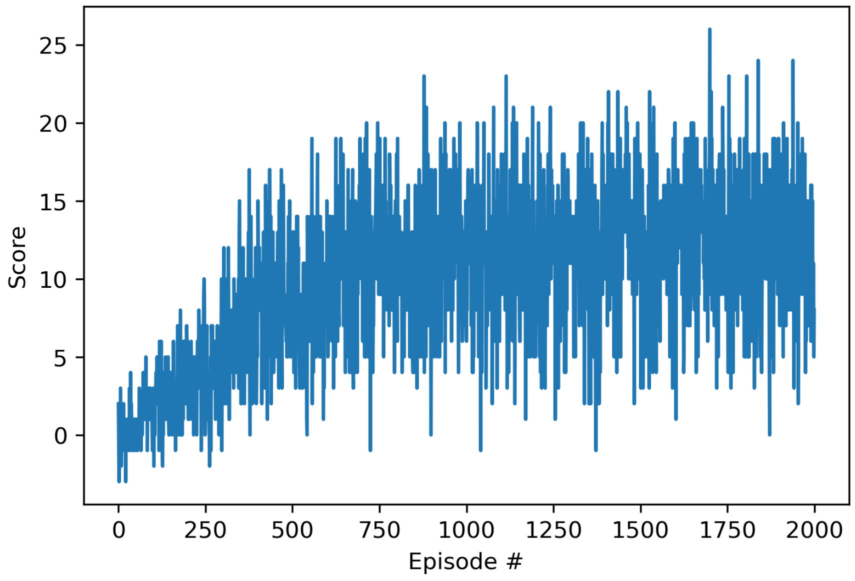

Figure 1: Reward plot of DQN

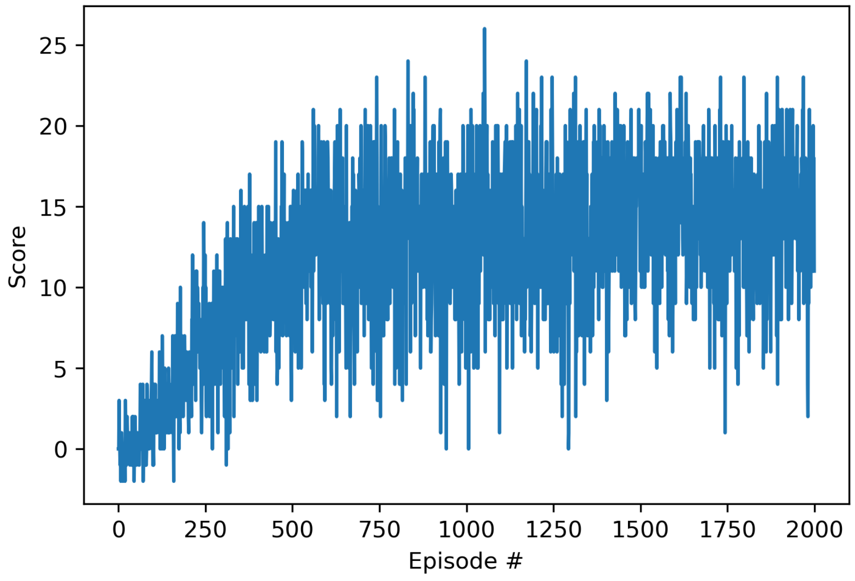

Figure 2: Reward plot of double DQN

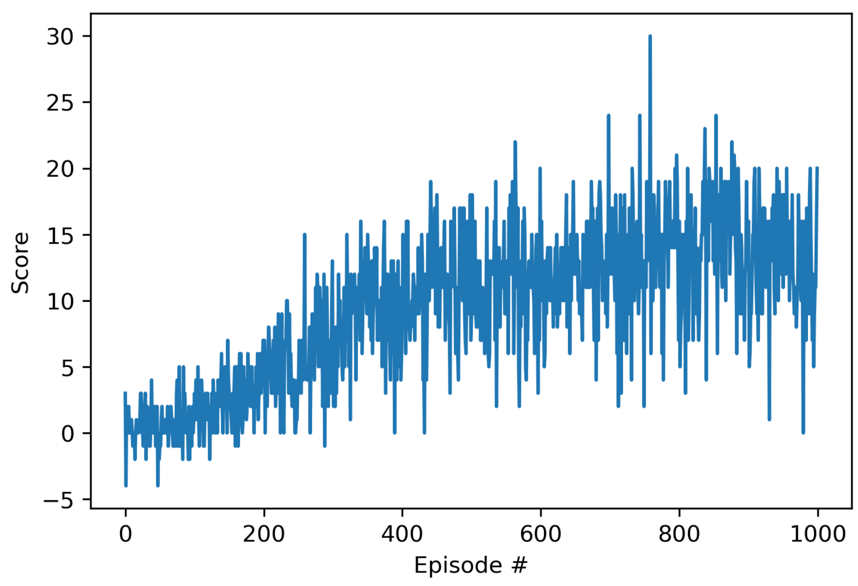

Figure 3: Reward plot of DQN with prioritized experience replay

- Double DQN and prioritized experience replay make the learning much faster and achieve better performance in the end.
- The training time for prioritized experience replay is much longer than the other two algorithms, due to un-uniform sampling strategy, and the priorities of sampled transitions need to be updated each iteration.

4 Future Work

- For prioritized experience replay, a sum tree should be used to store the priorities corresponding to the stored transitions, for better computational efficiency.
- Other extension of DQN could further improve the performance. Examples are dueling DQN, rainbow, etc.
- Try train the agent using pixels directly.

References

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., *et al.* (2015). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529–533.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.