Deep Reinforcement Learning Nanodegree

Project 1 : Navigation

Chenbo Gu

1. Problem Introduction

The target of the project is to train an agent to navigate around a square world and collect yellow bananas as many as possible while avoiding the blue ones. An introduction of rewards, states, actions and the project target is listed as follows:

Reward:
Collecting a yellow banana: reward +1
Collecting a blue banana: reward -1
State:
The state space has 37 dimensions including the velocity of the agent and the position information of surrounding objects.
Action:
4 discrete actions available:
0 - move forward.
1 - move backward.
2 - turn left.
3 - turn right.
Target:
An average score of +13 over 100 consecutive episodes.

More detailed environment information can be found on the github link provided by Unity:
https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Examples.md

2. Training Algorithms

Deep Q-Learning Learning(DQN), double DQN and dual network for vanilla DQN are adopted for training. Some changes are made to the sample code provided by Udacity.

2.1 Vanilla DQN

I. Hyper-parameters

```
BUFFER_SIZE = int(1e5)      # replay buffer size
BATCH_SIZE = 64             # minibatch size
GAMMA = 0.99                # discount factor
TAU = 1e-3                  # for soft update of target parameters
LR = 5e-4                   # learning rate
UPDATE_EVERY = 4            # how often to update the network
n_episodes=2000
max_t=1000
eps_start=1.0, eps_end=0.01, eps_decay=0.995
```

II. Model Structure

state_size-->fc1-->ReLU-->fc2-->ReLU-->action_size

fc1_units=64

fc2_units=64

2.2 Double DQN

The parameters and the model structure are the same as those of Vanilla DQN.

The only difference lies in the learning strategy.

2.3 dual network architecture for vanilla DQN

I. Hyper-parameters

```
# buffer size adjusted from 1e5 to 2e5
BUFFER_SIZE = int(2e5)      # replay buffer size
BATCH_SIZE = 64             # minibatch size
GAMMA = 0.99                # discount factor
TAU = 1e-3                  # for soft update of target parameters
LR = 5e-4                   # learning rate

# update_every from 4 to 6
UPDATE_EVERY = 6            # how often to update the network
```

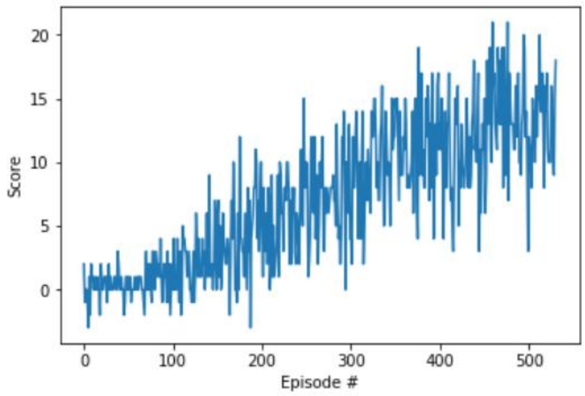
II. Model Structure

```
                            -->action_size(advantage funcion)-->ReLU
state_size-->fc1-->ReLU-->fc2-->ReLU                                 -->action_size
                            -->1(state-value function)-->ReLU
```

3. Results

3.1 Deep Q-Learning Learning(DQN)

```
Episode 100     Average Score: 0.53
Episode 200     Average Score: 3.17
Episode 300     Average Score: 6.48
Episode 400     Average Score: 10.44
Episode 500     Average Score: 12.20
Episode 531     Average Score: 13.01
Environment solved in 431 episodes!     Average Score: 13.01
```

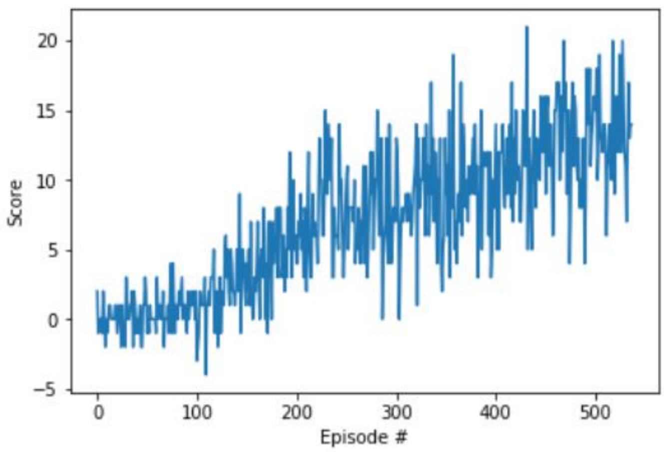


Training

Validation score : 21.0

3.2 double DQN

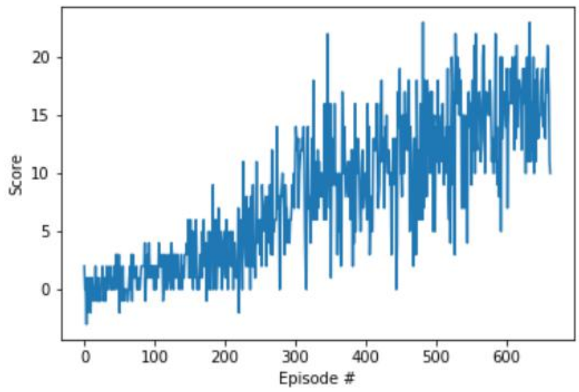
```
Episode 100     Average Score: 0.44
Episode 200     Average Score: 3.32
Episode 300     Average Score: 7.57
Episode 400     Average Score: 9.15
Episode 500     Average Score: 12.06
Episode 537     Average Score: 13.00
Environment solved in 437 episodes!     Average Score: 13.00
```



Validation score : 14.0

3.3 dual network architecture for vanilla DQN

```
Episode 100     Average Score: 0.68
Episode 200     Average Score: 2.22
Episode 300     Average Score: 5.15
Episode 400     Average Score: 9.49
Episode 500     Average Score: 11.35
Episode 564     Average Score: 13.02
Environment solved in 564 episodes!     Average Score: 13.02
Episode 600     Average Score: 13.91
Episode 664     Average Score: 15.33
```



Validation score : 24.0

3.4 Discussion

As we can see, all the three algorithms succeeded in solving the task in 600 episodes. It's hard to tell which one is better. Theoretically speaking, double DQN and dual network architecture for vanilla DQN should have better performances than vanilla DQN. However, it is not clear in this example. This may be caused by the following reasons:

I. This Navigation task itself is not a really challenging one.

II. The task requirement score>= +13.0 is not that difficult.

It is possible that the prons and cons of different algorithms are not obvious based on the first two reasons.

III. The performance depends highly on the parameters.

Personally speaking, the third reason may be the most significant one since my parameters are not well tuned.

4. Future Work

There is still much to do. For existed models, parameters should need tuning to achieve better performance. As for algorithms, I am still working on Prioritized Experience Replay and Rainbow. I will keep updating the repository.