Udacity Navigation Project

1.Learning Algorithm:

1.1.Introduction:

This project was solved using the deep Q networks method, which is a value based methods. The DQN main idea is to take a state as input and then returns action values for each possible action.

The DQN used for this purpose is based uses two key concepts:
- **Experience Replay**
- **Fixed Q-targets**

1.2.Experience Replay:

The experience replay technique is used to address the problem of high correlation that can be present in a sequence of experience tuples. Actually, the naive (or vanilla) DQN runs the risk of getting swayed by this correlation.

So the solution for this problem is to keep track of a replay buffer and use experience replay to sample from this buffer at random in order to prevent actions' values from oscillating or diverging.

Besides, the experience replay allows to learn more from individual tuples multiple times and recall rare occurrences, which means that we're using in a better way our experience !

1.3.Fixed Q-targets:

This technique is used to address another problem in the vanilla DQN which is the problem of **moving targets**. Actually, the TD target from the update rule of network weights is dependent on our approximation (current weights). And this can lead to harmful correlations!

The solution here is to decouple the TD target from weights' update. For this we will need two networks with identical architectures:

- **Primary Q-network**: its weights will be updated at every learning step
- **Target Q-network**: weights won't be updated at every learning step. The frequency of update is a hyperparameter.

2.Hyperparameters:

This is the list of hyperparams used along with their values and definitions:

- **BUFFER_SIZE**: indicates the replay buffer size for experience replay, value used 100 000
- **BATCH_SIZE:** Minibatch size for training, value used 64
- **GAMMA:** Discount rate, value used 0.99
- **TAU**: soft update of target parameters, value used 0.001
- **LR**: Learning rate, value used 0.0005
- **UPDATE_EVERY**: Frequency of updating the network, value used 4

3.Network Architecture:

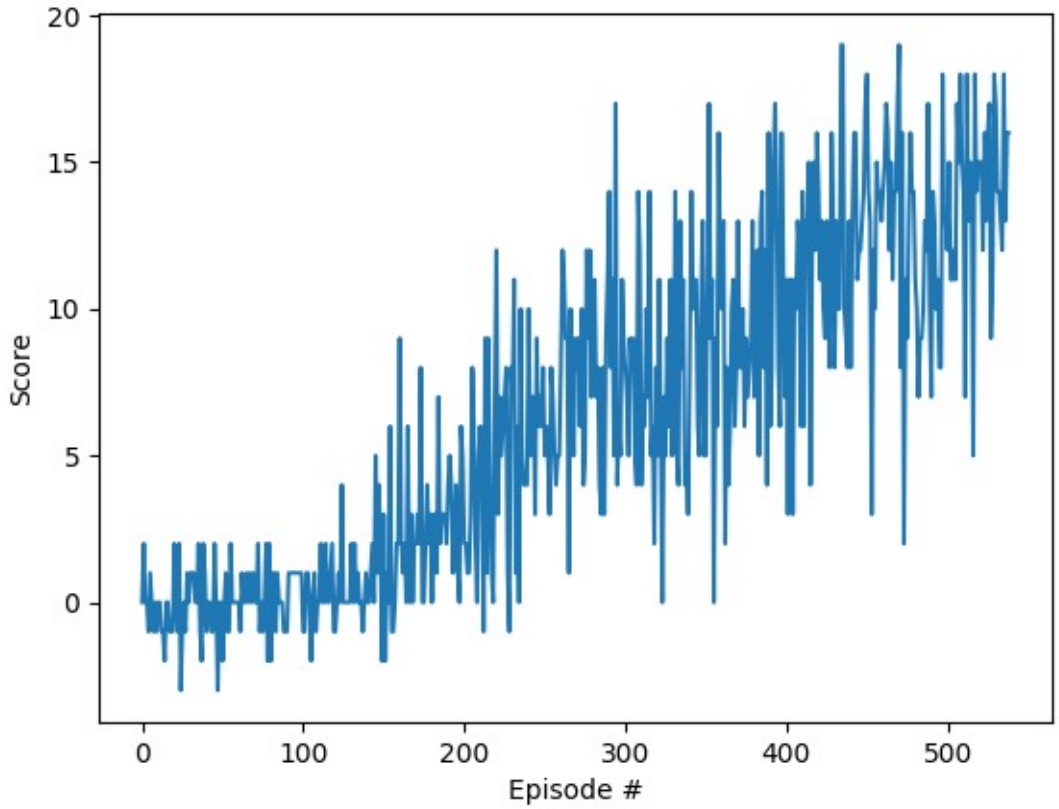
- Input layer: Fully connected layer with 37 units (state size is 37)
- 1st Hidden layer: Fully connected layer of 128 units, with relu activation
- 2nd Hidden layer: Fully connected layer of 128 units, with relu activation
- Output layer: Fully connected layer of 4 units (action size is 4)

Every unit of the output layer represent the action value of corresponding action.

4.Results:

The figure below shows the score evolution regarding the number of episodes done by the agent. As shown here, the problem was solved in fewer episodes than 600.

PS: To solve the problem, the agent must get an average score of +13 over 100 consecutive episodes.



5.Ideas for future improvements:

The deep Q-learning has several varieties that showed better performance than vanilla Q-learning or even Q-learning with only experience replay and fixed Q-targets that we used here.

Among these improvements, we can mention double DQN, dueling DQN or prioritized experience replay.