**Navigation Project Report**

*By Kevin Lee*

*Sep 3, 2018*

**Learning Algorithm**

This project uses a deep Q network to train an agent to pick bananas. A deep Q network algorithm consists of two main parts :

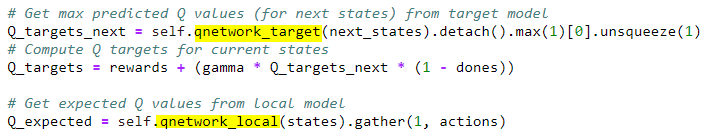
1. Re-inforcement learning
2. Deep Neural Network

The deep neural network is used as a universal function approximator. The main RL algorithm navigates the environment and feed to the DNN experience tuples that contain current state, action and next state as input. Then, the DNN learns to produce the appropriate next actions by trying to maximize future rewards.

There is another important component of the algorithm, which is the replay buffer. Instead of feeding the experience tuples directly in to the DNN as they are generated from the environment, the tuples are instead temporarily stored in a replay buffer. Only after certain number of tuples have accumulated in the replay buffer, learning will be carried out.

To learn, a random batch of experience tuples will be extracted from the replay buffer. By using a batch of random samples from the replay buffer, the correlation among the input tuples which were generated in sequence will be broken. That improves convergence. In addition, depending on the application, some data might be more important than others. Therefore, we could potentially vary sampling probability according to the importance of the data.

There is another critical component of the learning algorithm. As shown below, the Q\_targets and Q\_expected are kept as two separate sets. If it is not done this way, oscillations will happen while the neural network tries to learn and the results will become unpredictable.



**Hyper Parameters**

There are many parameters that can be tuned. Examples are :

BUFFER\_SIZE - replay buffer size

BATCH\_SIZE - minibatch size

GAMMA - discount factor

LR - learning rate

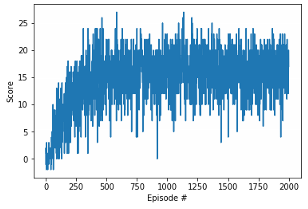
UPDATE\_EVERY - how often to update the network

EPS - Epsilon

I tried varying buffer size, batch size, learning rate and update\_every but found that I couldn’t improve the results very much. Then I tried changing epsilon.

Instead of a linear decay of epsilon, I tried starting with a slow decay. That means letting more random learning in the beginning. I started epsilon at 0.7. And then decay it aggressively, ie multiplying by 0.7 every 50 episodes. Then I achieved very good results. At about 1000 episodes, I could hit average score between 16 - 17. As shown by the graph below, the score actually hit a high and plateaud after about 600 to 700 episodes.

|  |  |
| --- | --- |
|  |  |



**Other Results**

Episode 
Episode 
Episode 
Episode 
lee 
2ee 
3ee 
313 
Average 
Average 
Average 
Average 
Score : 
Score : 
Score : 
Score : 
4.19 
8.27 
12 .38 
13. e3 
Environment solved in 213 episodes! 
15 
5 
o 
Average Score: 
250 
13. e3 
— int(1e5) 
BUFFER SIZE - 
BATCH SIZE = 64 
GAMMA = e.99 
TAU = 12-3 
UPDATE EVERY = 4 
replay buffer size 
minibatch size 
discount factor 
for soft update of target parameters 
Learning rate 
how often to update the network 
so 
200 
Episode # 

Episode lee 
Episode 2ee 
Episode 298 
Average Score: 3.15 
Average Score: 7.77 
Average Score: 13 . 04 
Envi ronment solved in 198 episodes! 
Average Score: 13 . 04 
20 
i 10 
O 
— int(1e5) 
BUFFER SIZE - 
BATCH SIZE = 64 
GAMMA = e.99 
TAU = 12-3 
UPDATE EVERY = 4 
if L episode X 10B e: 
print('hrEpisode 
counter | 
replay buffer size 
minibatch size 
discount factor 
for soft update of target parameters 
Learning rate 
how often to update the network 
Score: 
. sode, 
np. mean( scores_wi ) 
eps_decay eps_decay_set • counter 
50 
Episode # 
250 

Episode 
Episode 
Episode 
Episode 
12 .29 
Episode 
14.71 
Episode 
15 . 29 
Episode 
15 .63 
Episode 
Episode 
16.11 
Episode 
16 .63 
25 
5 
lee 
2ee 
3ee 
aee 
see 
see 
7ee 
8ee 
gee 
leee 
Average 
Average 
Average 
Average 
Average 
Average 
Average 
Average 
Average 
Average 
200 
Score : 
Score : 
Score : 
Score : 
Score : 
Score : 
Score : 
Score : 
Score : 
Score : 
Episode # 
2.11 
5.46 
8.81 
15 . 
77 
counter +_ 
eps_decay = 
eps_decay 
eps_decay_set 
set—e . '35 
counter 

**Model Architecture**

|  |  |
| --- | --- |
| Image result for deep neural network  *Just for illustrating the network architecture*  *Not the exact number of units for each layer* |  |

The deep neural network model used in this project closely resembles the graph above, although the actualy number of units are not fully drawn by the graph. The network is fully connected.

There are a total of two hidden layers with 64 units each and an output layer with 4 units. The activation functions used is the RELU.

**Next Steps**

Further steps can be taken to potentially improve the performance of the agent. These steps are beyond the scope of current project.

The firs thing I would try is to vary the architecture of the DNN. I would try deepening as well as widening the network.

The next thing I would try would probably be double DQN, dueling DQN, or prioritized experience replay. Some of these methods might potentially improve the performance of the agent.