Navigation Project – Work description

Learning Algorithm

The learning algorithm is the Double DQN with Priority Experience replay.

Double DQN – this algorithm is extension to standard Fixed Target DQN. The fixed target DQN actually stabilize the learning process via separation the target network and fixed it for defined number of iterations. This brings the idea of separation the learning network and the network which calculates the target value. Therefor the td-error calculation is more stable and it is not chasing itself. The Double DQN extend this idea by combine the either action. As the target network is some steps back, at the begging of learning can make a mistake by choosing wrong action. This is where double DQN comes in, the action is defined by current network, but value function is still taken from target network.

Priority Experience – I choose this algorithm because this looks have the significant improvement in learning process. The idea makes a more sense to me. The idea to learn from "highest" different between expectation and reality first and not just randomly select some states is really wise.

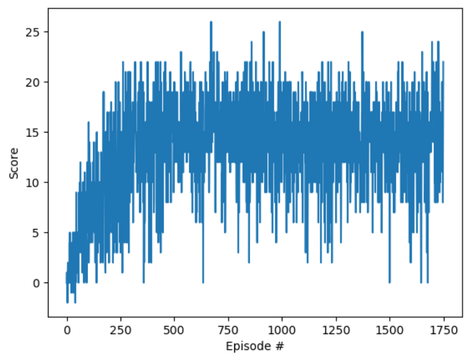
Neural network – I used only simple and small neural network, as this looks more stable. The network contains one hidden layer and 64 neurons. I also try the 4 hidden layers with 128 neurons, but this comes in unstable and very-very slow learning. After 250 episodes, there was less knowledge than after 80 episodes with small network. I either try to improve the network separately just by 2000 random state sample with random sample actions. I try different optimizer, different size of network. This just shows that with this setup it was blind-way. The network learns to choose only "0" reward. Because this was the most cases in random states. Which at the end is logic, and network does not reflect to very occasionally "1" or "-1" values.

Hyper param

The hyper-parameters

I choose
BUFFER_SIZE = 80000 # replay buffer size
BATCH_SIZE = 32 # 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
EPS_START= 1
EPS_MIN = 0.001
EPS_DECAY=0.99991

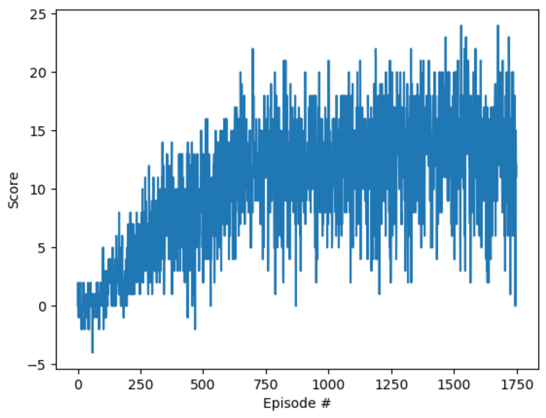
The environment was resolve in 340 episode and the final score looks like this.

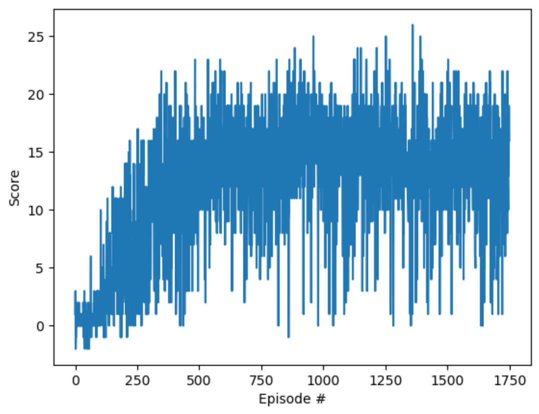


I test to change of BATCH_SIZE, UPDATE_EVERY, EPS_DECAY and let the rest fixed. The reason is that I realize the learning is very sensitive to parameters. And setup one of the parameters with wrong value, can lead into wrong assumption, that either something is wrong with code, and there must be error in algorithm. Because as the learning process does not convert, or convert very slow, or bouncing around 0. This happened to me and I start to find the issue in algorithm instead of playing with hyper-parameters. The best I found are placed upper.
To play with epsilon and increase the decay looks increasing the learning process, but brings on other hand more stable situations in later episodes. The random at early stages ensures to avoid the loosing later.

Epsilon

Compare the decay = 0.99999 on left with 0.9995 on right.

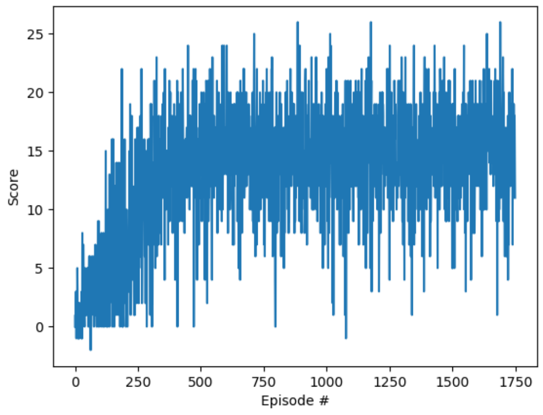


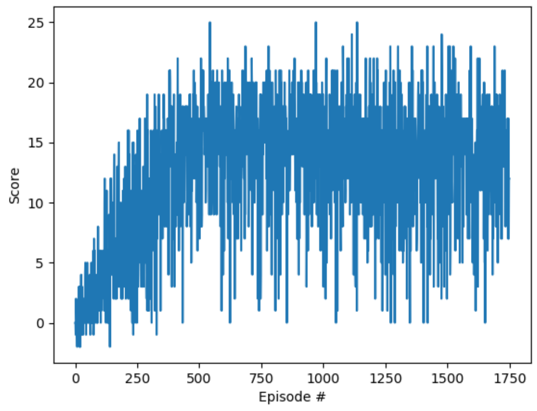


Batch size

It looks the batch_size has smaller impact than epsilon. Smaller batch_size looks to me more stable and have smaller deviation between episodes.

Compare batch_size 32 on left and 86 on right

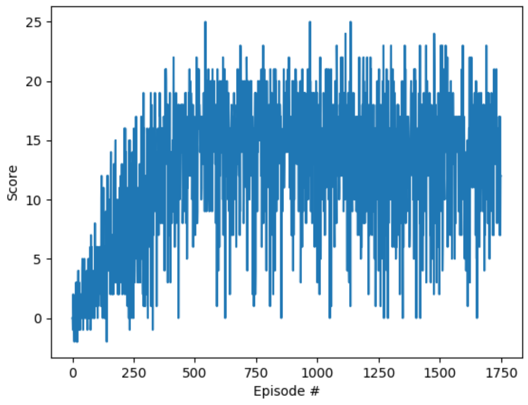


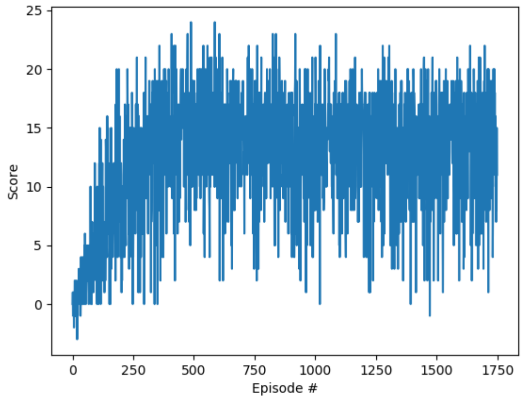


Update every – I compare the update every 3 and 4 steps.

It looks that update every 4 steps was little more stable. But this was tested with batch_size 86. Make this assumption more robust would require to test it with batch_size 32.

Update every 4 steps on left and every 3 steps on right





Ideas of future work.

From view of epsilon, it might be useful to just run about 50-100 episodes with full random actions. This might increase the knowledge of states for agent, and avoid to pick the "most common" action for unknow states in later phase. This would require to set the priority for random actions to priority min epsilon. And increase the buffer size. At the begging it will be some mix of random sampling and priority. Because the real priorities will be learned at later phases.

From view of batch size. I would try to perform strategy like “micro” batch. This means that sampling the higher number of experiences, but run the learning process in 2 or 3 “micro” batch. For example: sample 86 experiences, but divide into two groups and let the neural network been learn with two micro batches. As the neural network gets more chance to becomes stable because more update of weights can be performed in short time, and either the preciously gradient descent will be performed, because of smaller number of samples.

In same case the agent gets stuck between two blue bananas. Because the bananas are on left and right side, the agent was not able to step back as he learns to avoid this obstacle be moving right or left. I think this might be resolved by random sampling 50-100 episodes. This situation hopefully will not be affected by previous situation with just one blue banana. Or different but still combined approach can be do multiple steps not just TD-0. This might lead to eat the blue banana because future reward in next steps would be higher than just one step ahead.