Report - Project Navigation - Udacity Nanodegree: Deep Reinforcement Learning

Learning Algorithm:

Neural Network

Prior to starting the learning algorithm a Neural Network Architecture is set up. The NN is comprised of as many inputs as we have states in the RL problem (here 37). Then we have 2 hidden layers both with 64 nodes. Finally the output layer consists of as many nodes as there are actions in the RL model(here 4).

Algorithm

The learning algorithm works in the following way:

1. Reset the environment and determine a first state
2. Choose action according to what has been learned so far, with epsilon greedy
3. Make a step in the environment, gather reward and next state
    a. Save the experience in the replay buffer
    b. Every 4 steps - trigger a the **learning process**
4. If the episode is finished or you reached the max number of timesteps, repeat steps 1-4
5. If you reached the wanted number of episodes, finish

learning process

1. Get a batch of experiences from the replay memory.
   Those consist of a tuple (state, action, reward, next_state, done), with “done” being a boolean information on whether the episode has been finished
2. With this data we need to compute 2 values. One is the **target value** which is computed according to the Q-Value update formula. The other is the **expected value** which is simply “read” from the current values from the target network.
3. Compute the loss from those two values
4. Do a backward step which propagates the loss through the network
5. Do a optimizer step which adjusts the weights to the according to the computed gradient
6. Finally update the target network

Parameters

Batch Size: 64
(Size of tuples loaded from the replay buffer in a learning step)

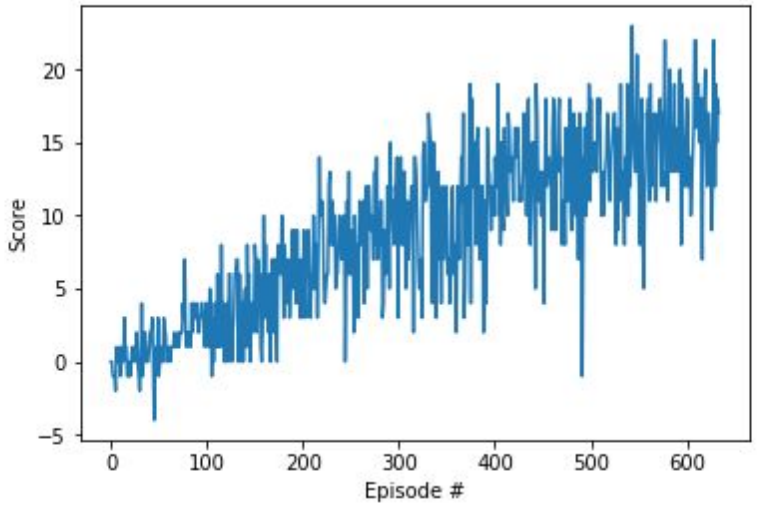
Gamma: 0,99

(Discount value for the Q-Update. This is the scalar that lowers the influence of the future value compared to the current value)

UPDATE_EVERY: 4 timesteps
(Determines how often to update the network as opposed to just adding the experience to the replay buffer)

Plot of Rewards:

```
Episode 100     Average Score: 1.14
Episode 200     Average Score: 4.20
Episode 300     Average Score: 8.16
Episode 400     Average Score: 10.04
Episode 500     Average Score: 12.81
Episode 600     Average Score: 14.41
Episode 634     Average Score: 15.01
Environment solved in 534 episodes!     Average Score: 15.01
```



(to keep the project-code simpler the code which is responsible for plotting was removed after the the graph was created)

Ideas for Future Work:

As is suggested in the project description another approach to solving this problem might be working with the raw pixel data and try to solve the problem with a convolutional Neural Network. In this case we would not feed the state values from the environment to the network, but would have to determine the next pixel values after a state has changed and give those to the CNN.

Another approach to change the outcome of the computation might be changing the parameters of the Network. We could try adding additional hidden layers or change the number of nodes in the layers. However is we reduce the size of the nodes this

might result in to few details to solve the task and adding more might bear the danger of overfitting the network.