Report - Project Continuous Control- Udacity Nanodegree: Deep Reinforcement Learning

Learning Algorithm:

Neural Network

Prior to starting the learning algorithm a Neural Network Architecture is set up.
To use the Actor-Critic Method we set up a seperate Net for the Actor and the Critic. Both Nets have two hidden Layers, with the first Layer having 400 nodes and the second 300 nodes. Both Nets get the states as input.
The biggest difference in the NNets can be observed in the Output Layers, where the Actor has as many output layers as there are actions available, to represent the probability distribution over the actions, and the Critic has only one node in the output layer, since it is supposed to output the state-value

Algorithm (DDPG)

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. If enough samples available - trigger 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

Since the technicalities of the learning process very much resemble the DQN Algorithm which was used and described in detail in the first project, we focus on the actor-critic aspects of the algorithm in this report.

The actor-critic method is applied for this problem to reduce **bias** and **variance**, which would otherwise cause problems in the learning stability in continuous control problems like this one. The method is implemented in the following way:

1. Get an **action a** from the current state of the actor-Net
2. Collect the **reward r** and next **state s'** from the environment taking action **a**
3. Train the critic-Net at **V(s, theta)** with the collected values in **r + gamma * V(s', theta)**
4. With the updated **V(s, theta)** train the actor-Net

Parameters

Batch Size: 128
(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)

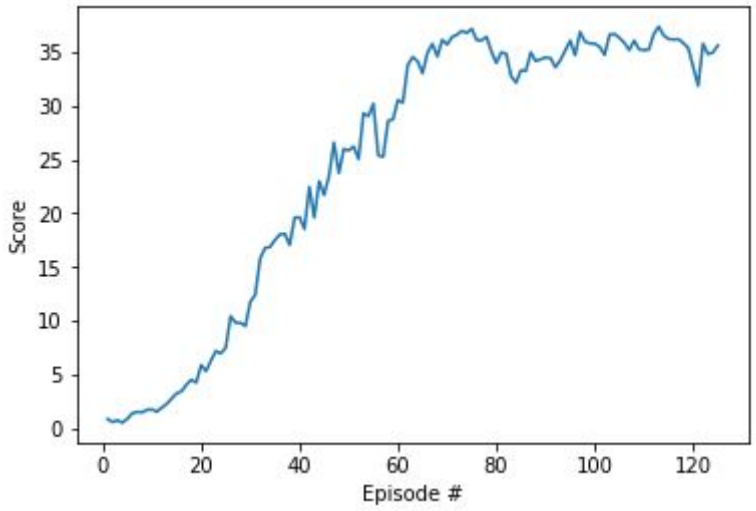
LR-Actor: 1/1000
(Learning-Rate actor)

LR-Critic: 1/100
(Learning-Rate actor)

Tau: 1/100
(Soft-Update Parameter for the interpolation of the target net)

Plot of Rewards:

```
Episode 25      Average Score: 2.86     Score: 8.009    2019-02-12 23:12:27.893683
Episode 50      Average Score: 6.87     Score: 12.442   2019-02-12 23:30:19.983394
Episode 75      Average Score: 9.94     Score: 16.791   2019-02-12 23:48:19.395525
Episode 100     Average Score: 13.31    Score: 31.758   2019-02-13 00:06:54.237313
Episode 125     Average Score: 21.29    Score: 36.249   2019-02-13 00:25:36.561079
Episode 150     Average Score: 27.34    Score: 36.952   2019-02-13 00:44:25.441220
Episode 163     Average Score: 30.06    Score: 36.806   2019-02-13 00:54:16.550299
```



(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:

Taking an idea from looking at the next chapter, it might be beneficial if the agents could gain some experience from watching another agent act. So maybe if a new agent had access to the actions of some already learned agent it could try to resemble his actions. In this specific scenario however its not clear that this might be really beneficial since it seems like the agent then could only use the other agents actions as well since there is overarching goal that they might achieve. This could be more useful if the agent could only observe certain parameters of the other agent, like how close his arm is to the target for example.

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 as 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. In some tests adding some minor changes to the network parameters did not have a significant influence on the learning results.