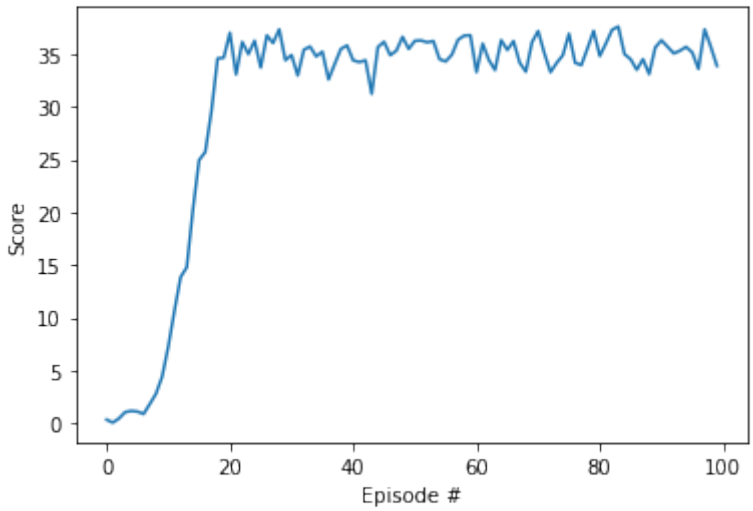
Report

In my implementation I used the code for a DDPG-agent, but modified it, such that he can learn from all 20 different environments.
The jupyter-notebook „Run.ipynb“is the main file, where we load the environment, set up the agent and are able to monitor the progress. When the agent reaches an average reward of 30 over the 100 episodes, it stops, saves the model and returns a plot of the learning curve of the agent.
The agent itself uses an actor network to compute the next action. We used tanh-activations at the end to get numbers between -1 and +1. To let the agent explore the environment, we always add a small error sampled from a normal distribution with $\mu = 0$ and $\sigma = \epsilon$. We start with an $\epsilon = 1.$ and use an exponential decay with decay rate 0.95. We use a lower bound for ϵ: $\epsilon_{min} = 0.05$.
Additionally, we constructed a critic to rank the effect of the action and compute the Q-values. The agent has a replay puffer of size 10000, where it stores the different experiences it makes. We update the networks every 4 steps. The other hyperparameters are:

$$\gamma = 0.99\,, \qquad \tau = 0.001\,,$$

The actor takes as input the current state. It has 4 dense-layer, 3 layers with 64 neurons, and the output-layer has 4 neurons. The hidden-layers have ReLu-activations, while the output layer has a tanh-activation. The critic takes as input the state and the action of the step. It has 3 dense-layer, 2 layers with 64 neurons, and the output-layer has 1 neurons. The hidden-layers have ReLu-activations. We train our networks with a learning rate of $5 * 10^{-5}$ with Adam optimizer and a batch size of 64.
As a result we get for the average reward the following plot:



As we can see is our agent able to solve the environment within 0 steps. After 19 steps it

performs well enough to get as reward values above 30.

For the future,we can try to simplify the network. Another possibility is to use A2C or A3C implementations to solve the problem.