DESCRIPTION OF THE ALGORITHM

The algorithm in the Continuous Control Jupyter Notebook is the DDPG algorithm of Lillicrap et al.

It is an off-policy algorithm for deep reinforcement learning. It is one of the first algorithms to allow to operate in environments with continuous actions. In order to make this possible it uses the Actor Critic paradigm. In other word it contains 2 nets, one that acts (the Policy) taking a state s and spitting out continuous numbers for each action a, and another one (the Critic) that discriminate the policy, i.e. the value of the action a under state s.

More specifically the Policy net is a net with input size the input of the environment and output size the the number of actions. To the output of the Policy it is usually added a noise in order to facilitate exploration (this because in this context we do not have an epsilon greedy possibility). The Critic is a net with input size the size of the observation space of the environment plus the size of the action space (this is in practice slightly different in order to improve the performance of the net).

During the training the Policy plays, adding its experience to a replay buffer and with a certain frequency we batch experience from the buffer, passing this experience to the Critic. While the Critic improves its discrimination in a DQN style, also the Policy improves, leading to convergence in a certain amount of time.

DESCRIPTION OF THE NETS

The Policy net contains three linear layers with added ReLU functions. The hidden sizes contains respectively 256 and 128 nodes. The input size is the size of the observation space the output size is the size of the action space.

The Critic net contains three layers. The first layer has input size the size of the observation space and hidden size 256. The second layer has input size 256 plus the size of the action space and output size 128 (we concatenate the action space and the previous hidden nodes and feed it to the second layer. We do this, instead of feeding action and state at the same time at the first layer in order to prevent the state space, which it is usually much bigger than the action space) of having a predominant factor). We then use a third layer with dimensions 128 and 1.

DESCRIPTION OF THE LEARNING PARAMETERS

$BUFFER\ SIZE = int(1e6)$
$BATCH\ SIZE = 128$
$GAMMA = 0.99$
$TAU = 1e-3$
$LR\ ACTOR = 1e-4$
$LR\ CRITIC = 1e-4$
$WEIGHT\ DECAY = 0$

$UPDATE\ FREQUENCY = 1$
$NUM\ UPDATES = 1$

Most of the parameters are self explanatory. Let's just indulge over
$UPDATE\ FREQUENCY$: which decides when update Policy and Critic. Which in our case is every time we collect a new piece of information (a tuple $(s, a, r, is\ done)$) $NUM\ UPDATES$: which decides how many times we update the nets.

TABLES

The table of the plotting rewards is included inside the Jupyter Notebook. Notice it contains just the last 100 episodes since it was highly computationally demanding to solve the environment and I trained the nets over multiple times. The number of episodes to solve the environment varies between 900 episodes and 1100 episodes.

FUTURE IDEAS

The most straightforward idea is to implement D4PG. One has to add prioritized experience replay and a distributed factor to the algorithm. Also, the learning part depends on the distributional algorithm of Bellamere.
Easier ideas are to investigate different architectures for nets and different hyperparameters which might be more optimal than ours.