Continuous Control - Reachers

1 Framework

1.1 Environment objective

There are 20 parallel agents that stand for double-joint robotics arms which objective is to learn to follow a target as long as possible, for such purpose each agent receives a positive reward of +0.1. To tackle this task a multi-agent version of the Deep Deterministic Policy Gradient (DDPG) is used.

1.2 DDPG

The model was introduced as an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces[1]. The algorithm uses a couple of neural networks the first called actor it's used to learn the best action in a continuous action space, and then uses this estimate to train a second network called critic to learn the optimal action-value function. The critic is used to give more stability to the policy gradient method which tend to have high variance. The following pseudo-code stands for the DDPG algorithm introduced in the original paper:

1.2.1 DDPG

Algorithm 1 DDPG algorithm[1]

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$with weight θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for $episode = 1\,M$ **do**
    Initialize a random process N for action exploration.
    Receive initial observation state s_1
    for $t = 1\,T$ **do**
        Select action $a_t = \mu(s_t \mid \theta^\mu) + N_t$ according to the current policy and exploration noise.
        Execute action a_t in emulator and observe reward r_t and s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample random minibatch of N transitions (s_j, a_j, r_t, s_{j+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid Q^{\mu'}) \mid \theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
        $\nabla_{\theta^\mu} J = \frac{1}{N}\sum_i \nabla_a Q(s_i, a_i \mid \theta^Q) \mid_{s=s_i, a=a_i} \nabla_{\theta^\mu}\mu(s \mid \theta^\mu) \mid_{s=s_i}$
        Upgrade the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

So the algorithm create an actor and critic-networks and and a copies of each one designed to be the target networks and be update only after certain number of steps have occurred using a soft update (it blends the local-networks with the targets-networks with a convex combination). Basically follows the line dictated by the DQN algorithm but using the actor-network to approximate the $argmax_a Q(s, a \mid \theta^Q)$n thus the critic network assess the return given the current action and state, and finally the actor network uses critic value to maximizes the return as a function of the best action.

2 DDPG implementation

At each step the environment returns the rewards and next states for each agents these are stored along with previous states and the taken actions in the replay buffer, the agent keep taking actions and repeating this process until after a certain number of steps a sample batch is taken from the buffer and the learning process starts. After compute the updates for each network the weight are propagated to the agents.

2.1 Hyper-parameters

Hyper-parameters		
Parameter	Value	Description
BUFFER_SIZE	100,000	replay buffer size
BATCH_SIZE	128	minibatch size
GAMMA	0.99	discount factor used in TD targets
TAU	.001	for soft update of target parameters
LR_ACTOR	.003	learning rate used by the optimizer algorithm of the actor neural network
LR_CRITIC	.003	learning rate used by the optimizer algorithm of the critic neural network
UPDATE_EVERY	1	how step are needed to update the networks
NUMBER_UPDATES	1	number of time the networks are updated in each leaning phase
HIDDEN_LAYER_1	64	number of nodes for the first fully connected layer in critic and actor's network
HIDDEN_LAYER_2	128	number of nodes for the second fully connected layer in critic and actor's network

Different number of layers were used to define the networks but after two layers the expected time to train the model augmented drastically, so a number of 2 layer were appropriate to train the agent. In the same fashion a different combinations of nodes in each layer were tried(all of them were powers of 2) and this was the fastest combination to trained the model that I observed.

2.1.1 Network's architecture

Actor Network		
Layers	Nodes	Activation function
Input layer	33	Rectifier Linear Unit
First hidden layer	64	Rectifier Linear Unit
Second hidden layer	128	Hyperbolic Tangent
Output layer	4	NA

Critic Network		
Layers	Nodes	Activation function
Input layer	33	Leaky Rectifier Linear Unit
First hidden layer	68	Leaky Rectifier Linear Unit
Second hidden layer	128	Identity
Output layer	1	NA

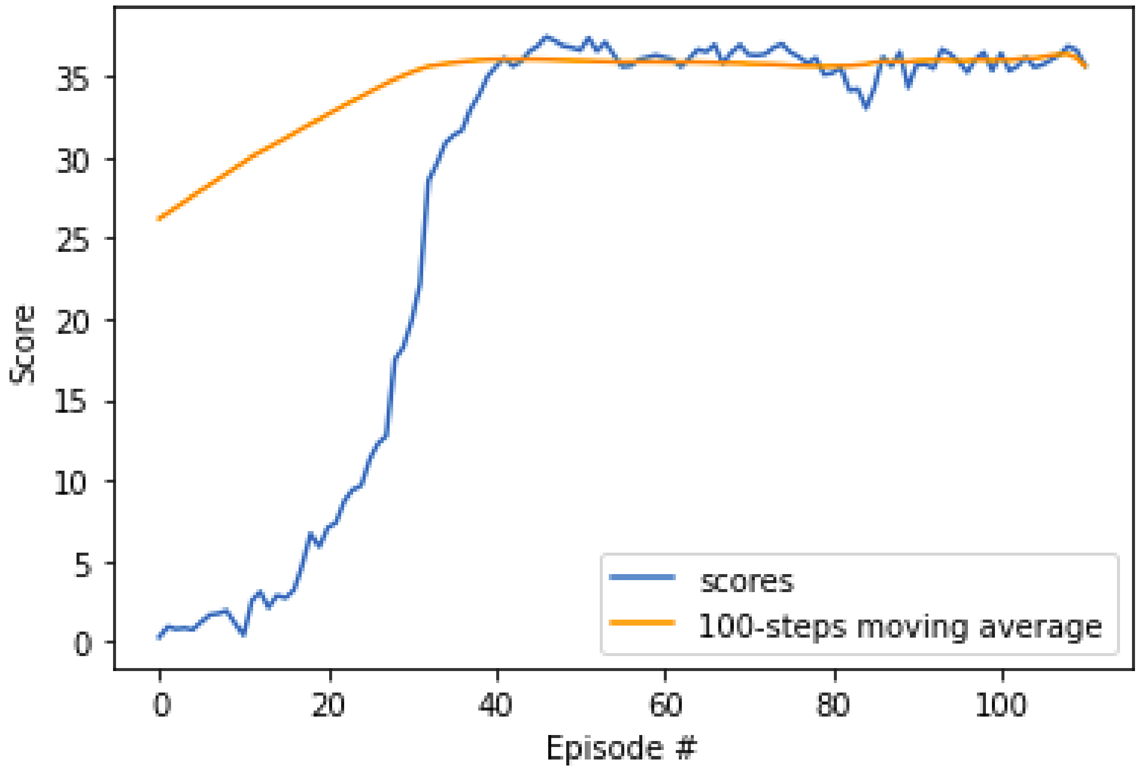
To the output of the actor network the noise coming from an Ornstein–Uhlenbeck process is added and then clipped between -1 and +1. Another thing to notice is after feeding the input layer of the critic-network with states, the output of that layer and the actions coming from the output of the actor-network are concatenated to finally feed the second layer of the critic-network.

3 Results

Following can be observed how the agents perform through the training phase, their average performance over a window of 100 hundreds episode and the moment they solved the environment. The next table show the result for the first 20 episodes.

DDPG Agents		
Episode	Score	Average Score
1	0.273000	26.174074
2	0.938500	26.535534
3	0.793500	26.879354
4	0.858000	27.227534
5	0.756000	27.580699
6	1.237000	27.928609
7	1.682500	28.273379
8	1.758500	28.616674
9	1.934000	28.962289
10	1.178500	29.311414
11	0.410500	29.665719
12	2.544000	30.017844
13	3.100000	30.295358
14	2.110500	30.572862
15	2.863500	30.866288
16	2.710500	31.157984
17	3.211000	31.457431
18	4.729000	31.757925
19	6.696500	32.048558
20	5.888500	32.324124

Environment solved in 11 episodes! with Average Score: 30.02.



4 Future Work

The following steps for this exercises would be:

Implement the D4PG[2] algorithm to the environment, the enhancements would be to change the the replay buffer per a prioritized replay buffer (PER), enable the use of n-steps bootstraping to compute the time difference targets(TD targets) and finally implement distributional critics network. The last step has to be studied further by me and it is perhaps the most difficult part in my point of view, in fact I've already worked on the implementation of D3PG but I have to test it and make the agents converge.

References

[1] Timothy P. Lillicrap, Jonathan J. Hunt, et al(2016). CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING. Google Deepmind. London, UK.

[2] Gabriel Barth-Maron , Matthew W. Hoffman, et al(2018). DISTRIBUTED DISTRIBUTIONAL DETERMINISTIC POLICY GRADIENTS. Google Deepmind. London, UK.