Project Reacher

1. Introduction

In this project there are 20 different agents where every agent as an arm with two joints. The action space consists of four continuous actions between -1 and +1 that applies torque to the joints. The goal of each agent is to keep their respective arm at a specific target zone.

Each agent gets a +0.1 reward for every time step that the agent can keep their arm within the goal zone.

The state space consists of 26 different states for each agent which consists of position, rotation, velocity and angular velocities.

The goal of the agents is to achieve a rolling mean of more than +30 over 100 episodes averaged over all the 20 agents.

In Figure 1 a visual of the Reacher environment is depicted with 10 different agents.

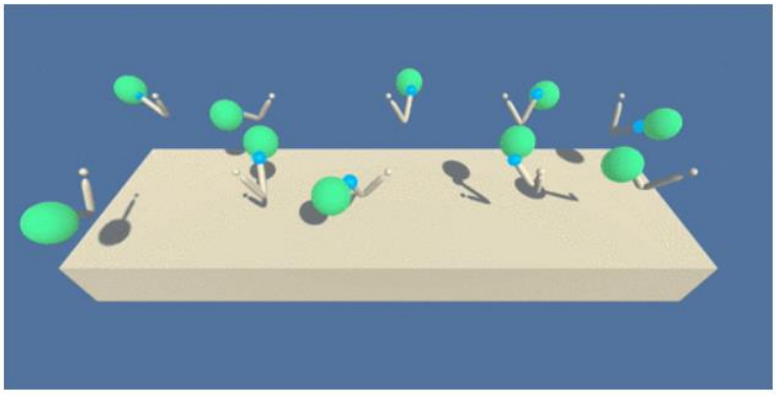

Figure 1: The Reacher environment with 10 different agents.

DDPG

In this project the Deep Deterministic Policy Gradient (DDPG) is used. The pseudocode for this algorithm is depicted in Figure 2. The algorithm consists of two networks one for the critic and one for the actor, where each network has a target and local network similar to the DQN with a target network.

The policy network also as a noise term to add some exploration to the actions taken. The noise is governed by the Ornstein-Uhlenbeck process for more details see this paper: https://arxiv.org/pdf/1509.02971.pdf

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 2: Pseudocode of DDPG, from https://spinningup.openai.com/en/latest/algorithms/ddpg.html

2. Method and parameters

The method used in this project is the Deep Deterministic Policy Gradient (DDPG). The code in this project is based on the Udacity's DDPG code used to solve the bipedal environment and it can be found in this repository.

Hyperparameters used and their respective values:

- Batch size: 256
- Gamma: 0.99 (discount factor)
- Learning rate for actor and critic network: 1e-4

- Tau: 0.001 (soft update blending factor between target and local network)
- Update episode frequency: 10 episodes
- Number of updates of the actor and critic networks: 20
- Buffer size: 1e5 (Size of replay buffer)
- Learning rate: 3e-4 (for both actor and critic network)
- Optimizer: Adam with default settings

Running parameters:

- Episodes: 400
- Time steps: Until termination
- Once we reach the target of a rolling average of +30 for the agents the learnings rates for the actor and the critic are divided by 10, the same thing is done for the tau, the blending factor.
- For every episode the factor that is multiplied with the action noise factor is reduced by a decay factor of 0.984. This noise multiplication factor is reduced to a min of 0.1.

2.1. Actor Network

Layers and units:

- Input layer: 26
- Hidden layer 1: 128
- Hidden layer 2: 64
- Output layer: 4

Action functions: Relu between all layers except the last where we a tanh to get a value between -1 and +1.

2.2. Critic Network

Layers and units:

- Input layer: 26
- Hidden layer 1: 128 + 4(actions)
- Hidden layer 2: 128
- Hidden layer 3: 64
- Output layer: 1

Activation functions: leaky relu for all layers except the last one that doesn't have any.

3. Results

In Figure 3 the results of the run with the above-mentioned configurations are depicted. We can see that the rewards per episode seems to get above the target after around 60 episodes. The rewards linger around the target line until about episode 175 and then increase to about 35 for about 50 episodes. Then we see a dip of the rewards down to about 30 again before it goes back up to around 35.

The rolling mean target is fulfilled at around 150 episodes then the rolling mean is slowly increasing up to around 35 until we stop training at episode 400.

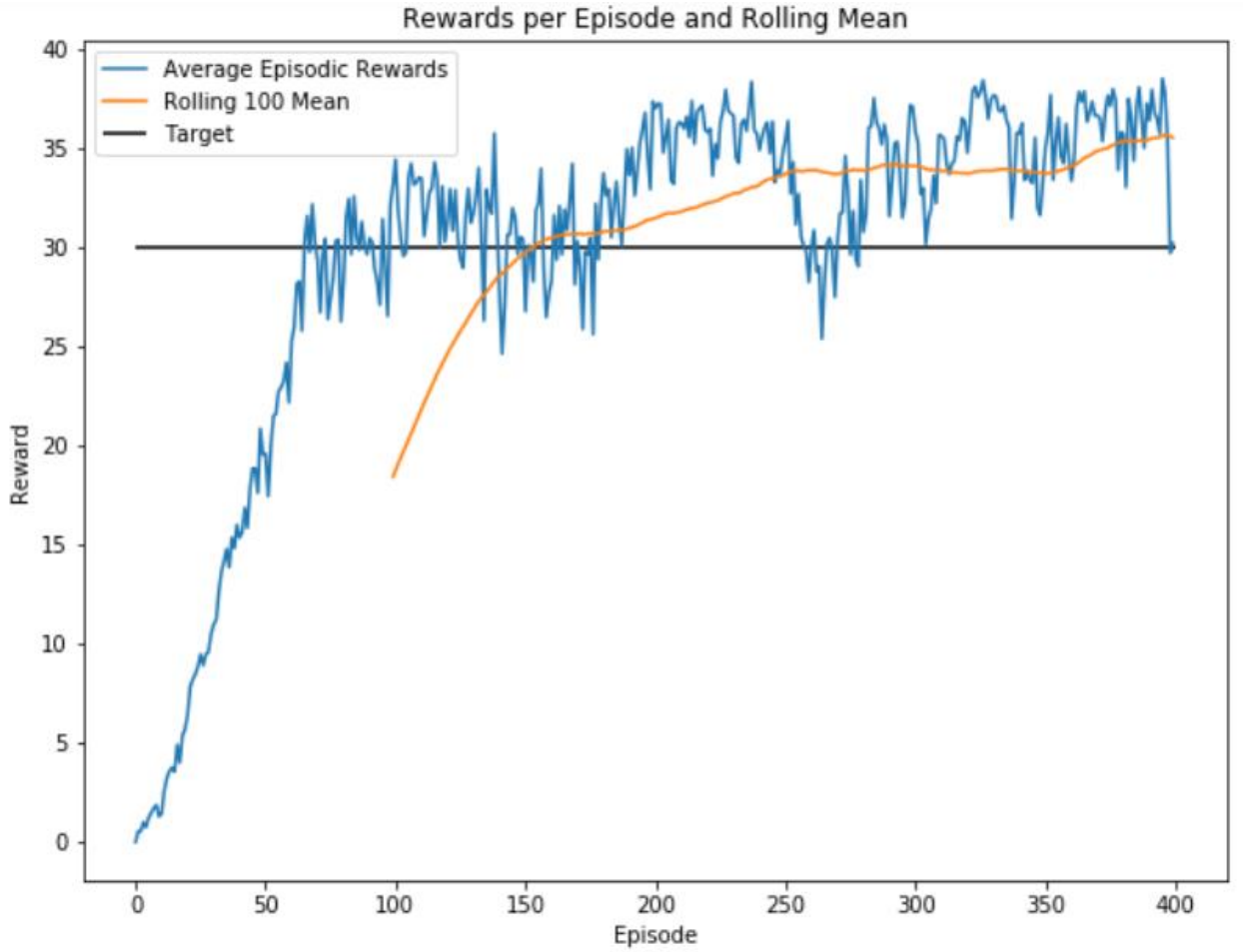

Figure 3: Rewards per episode and rolling mean over 100 compared to the target.

4. Discussions

Many different configurations of different hyperparameters were tried for both the 1 and the 20-agent system. It took me a long time before I started to get some idea about what parameters that work to accomplish the target.

Recommended steps to further investigate would be to both reduce the size of the network but also to increase it. It would be interesting to reduce the network size to get faster run times to easier find optimal settings for the hyperparameters. However, it would also be interesting to increase the network size to increase the capacity and see how large scores one could achieve. However, this would come with the cost of longer run times.

I would like to run the environment with a learning rate scheduler to have an adaptive time step dependent on the loss functions. This would help to reach the target goal faster and then hopefully stabilize the training thereafter.

In addition the update frequency, number of updates and the blending factor should also be somewhat adaptive dependent on the loss output of the networks.