Sapienza Università di Roma
Department of Computer, Control, and Management Engineering Antonio Ruberti
Artificial Intelligence and Robotics

Project

Course: Reinforcement Learning

Student: **Amila Sikalo**
Matricola: **1938032**

February 20, 2021

1. Introduction

This report is on algorithms of reinforcement learning developed for continuous environments. We will be discussing the implementation of TD3 and HER algorithms for navigating a robot manipulator, so it can reach a certain position. This environment is taken from an open AI gym library called fetch reach and the main purpose of this is to learn the optimal policy to navigate the robot, so it could reach these randomly given points in space.

2. Problem description

The fetch reach problem from an open AI gym consists of the robot manipulator with some degrees of freedom, that must move from a starting position to a certain goal position. This means that in order to succeed the robot must reach the final point that is as close as possible to the given goal point. If this final point is coinciding with the given goal point, then reach??? school and desired goals are the same and therefore the robot performs the task successfully. Otherwise, it is not successful. This is the metric that we will use in order to evaluate the performance of the learned policy. That means that in order to maximize the reward the robot must complete the task successfully in as many tries as it can. This will be evaluated using a metric that is given within this gym environment and it evaluates whether the desired goal position and reached goal position are the same. This will be better explained in results where we will evaluate the trained model with the precision of reaching the desired goal.

3. Algorithm description

In this part, we will discuss the background of these two algorithms TD3 and her, and we will see how the mathematical model was developed for these two algorithms. In the next part, we will discuss the implementation details for these two algorithms.

TD3 algorithm is one of the newer family of algorithms that were developed in order to solve continuous control problems. It builds on DPG algorithm or deterministic policy gradient algorithm, which is an actor critic method that uses value estimate that was learned previously to train a deterministic policy.
Another algorithm from this family is DDPG or delayed deterministic policy gradient which was developed in order to improve learning rate during the iterations because of this delayed learning. TD3 builds on the knowledge obtained with DDPG but introduces another delay that minimizes the approximation error that led to overestimated values of value function or policy function. This algorithm minimizes the effects on both active and critic models in order to reduce this effect. As mentioned before this is an actor-critic structure where we develop two neural networks that will learn an optimal policy for controlling the continuous system such as before mentioned fetch reach. The policy structure is known as the actor because it is used to select actions from the set of possible actions of the environment and model mainly robot in this case. An estimated value function is known as the critic because it values or criticizes the actions made by the actor learning, in this case, is off policy which means that we are sorry scratch that I am still not sure whether it is of policy on policy let me check did you TD3 algorithm works as follows: we first start with randomly assigned parameters to the actor in treating networks. We also define something called target networks which we will use to train the model. We also initial initialize a replay buffer for storing transitions already made during the learning process. Obviously, the replay buffer, in the beginning, will be an empty set. Now what we do is to select an action and we can add an exploration noise. Also, we must observe the reward we get from this action and observing the new state. This transition is then stored in a replay buffer because we already made that transition so we can learn from some experience that we gained during this learning process. Now we calculate actions, and policies and then we update critic networks by minimizing the error in learning policy. Now what differs from the standard algorithm is that we do not update target that works in every iteration but use some sort of delay so it means that every K, let us say iterations we will update the gradient of deterministic policy and then use that for updating target networks. This is further explained in pseudocode of TD3 algorithm:

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ
with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
  $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Hindsight experience replay (HER) is a sampling technique for learning from rewards that are sparse and binary. In this case, we have a reward in the sense that the robot reaches the given position, or it does not reach the given position, so it's either zero or one. This human-size experience replay is very good for systems environments in which rewards are sparse. Not every state transition produces a reward. The good thing with her is that it can work with any policy reinforcement learning algorithm and it is used as learning from previous mistakes strategy. So, the idea of HER is to store in replay buffer every transition that we had during some episodes. For example, in episode K we had transitions S, 0, S_1, up to S_t. Then we store every transition as $S_t S_{t+1}$, and not only with the original goal used for this episode K, but also with a subset of other goals. The idea is to not only use current episodes and current transitions for learning but also previous transitions as well. This is better explained in the pseudocode given in the following algorithm:

Algorithm 1 Hindsight Experience Replay (HER)

Given:
- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = 1, M **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T - 1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T - 1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

4. Implementation results

Parameters for Scenarios:

Scenario 1 Parameters:
n-epochs=30, n-cycles=5, n-batches=5, replay-k=4, lr-actor=0.001, lr-critic=0.001, gamma=0.99
Scenario 2 Parameters:
n-epochs=30, n-cycles=5, n-batches=5, replay-k=8
Scenario 3 Parameters:
n-epochs=30, n-cycles=10, n-batches=5
Scenario 4 Parameters:
n-epochs=30, n-cycles=5, n-batches=10
Scenario 5 Parameters:
n-epochs=30, n-cycles=5, n-batches=5, lr-actor=0.01, lr-critic=0.01
Scenario 6 Parameters:
n-epochs=30, n-cycles=5, n-batches=5, gamma=0.8

It is important to say that the Scenario 1 is the base scenario, and all other the following scenarios, only changes compared to the first scenario are listed.

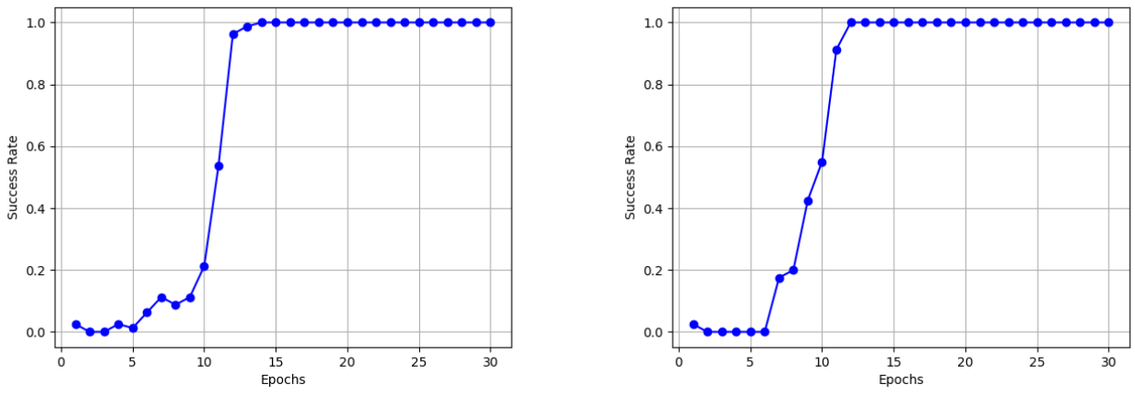

Figure 1: Scenario 1 and Scenario 2

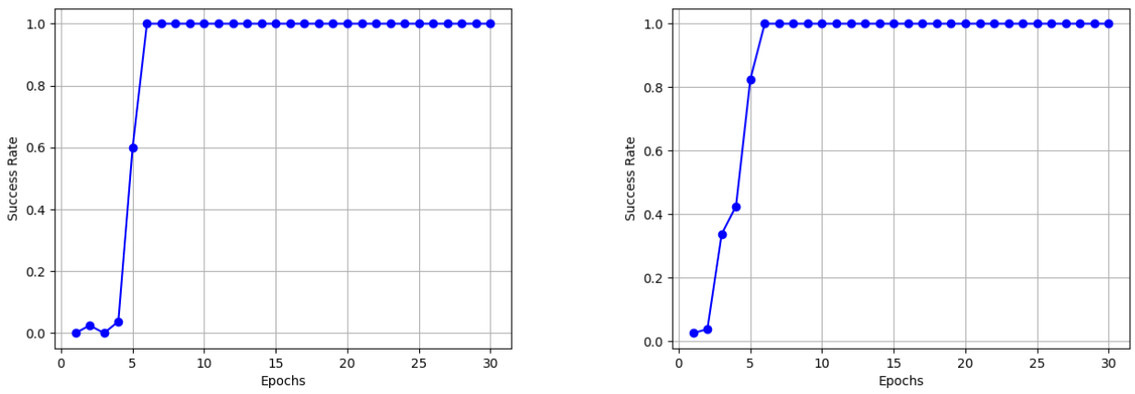

Figure 2: Scenario 3 and Scenario 4

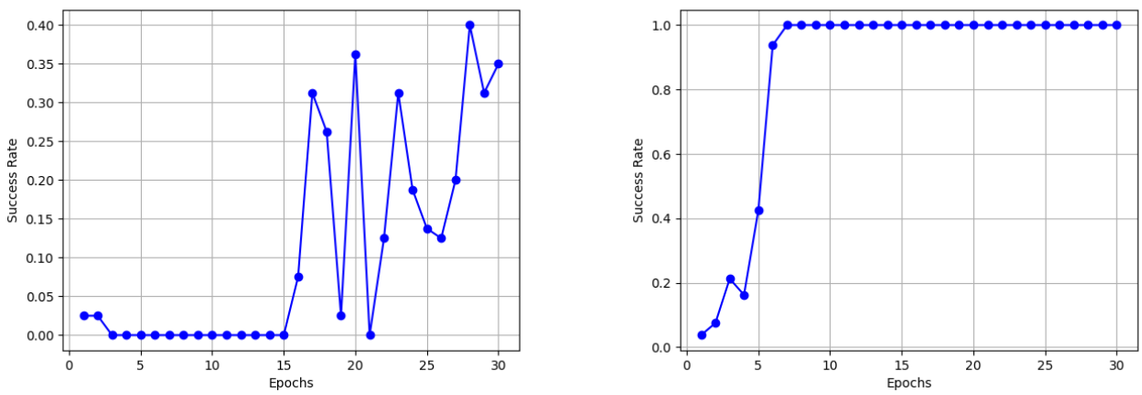

Figure 3: Scenario 5 and Scenario 6

5. Results

6. Conclusion