IMPLEMENTATION OF TD3+HER ALGORITHM TO SOLVE ROBOT PICK AND PLACE TASK

REINFORCEMENT LEARNING IMPLEMENTATION PROJECT REPORT

Gioele Migno - 1795826

2021/02/19

ABSTRACT

The project consists in the implementation of TD3 algorithm and the use of HER method in order to train a robot to perform pick and place tasks. The project development is composed of three parts, TD3 implementation test, TD3+HER test on the simplest robot environment and then the use of TD3+HER to solve the target task.

1.INTRODUCTION

1.1 ROBOT ENVIRONMENTS

The robot environments used are part of OpenAI Gym suite and they are based on Mujoco physics engine. In this project two different environments have been tested:

- ***FetchReach-v1***: The goal of the robot is to reach with its gripper a target position in a 3D space.
- ***FetchPickAndPlace-v1***: The goal is to grasp a box and move it to a target position in a 3D space, in some cases the target is on the table and in other cases in the air.

In both environments the robot is a robotic arm with rotation joints and two fingered parallel gripper.

1.1.1 OpenAI Gym Interface

The observation of the environment returned by OpenAi gym is described by a dictionary composed by three keys;

- ***observation***: Actual observation of the environment that contains information about the cartesian position and velocity of the robot's gripper and when present also the position, velocity and orientation of the object to move.
- ***achieved_goal***: Current goal reached by the agent.
- ***desidered_goal***: Goal that the agent must reach.

The action is a 4D vector that specifies the desired gripper movement in cartesian coordinates (3 components) and the opening of the gripper.

The rewards are sparse and binary, the agent obtains a reward -1 for each step and reward 0 when it is near to the goal, for example in the case of pick and place task, reward is 0 if the object is in the target position with a tolerance of 5cm. Each episode is composed of fifty steps.

1.2 DDPG

Deep Deterministic Policy Gradient is an algorithm that combines the two main types of model-free methods, Policy based (Policy Gradient) and Value based (Q-Learning), in order to obtain an off-policy algorithm applicable to environments with continuous action space. DDPG during the training learns both a policy and a Q-Function represented by an Actor and a Critic respectively.

Critic

The Critic is trained by sampling at each iteration a mini-batch of N transition <s, a, r, s', not_done> and using as loss function the MSE between the current Q-Function estimation (Q_critic) and the target y given by:

$$y = r + not\_done * \gamma * \max_{a'} Q_{critic}(s', a') \approx r + not\_done * \gamma * Q_{critic}(s', \pi(s'))$$

Since the computation of maximum value of the Q-Function at each iteration is not feasible, the target uses, as approximation, the Q-Function calculated in the next state s' and the action chosen by the Actor.

However, in practice, since both Actor and Critic are neural networks, in order to stabilize the training, it is necessary to use in the computation of the target y, two different networks called target networks: Q_critic_TARGET, and pi_TARGET. These networks have similar parameters of the current networks but with slow changes.

So, the loss function to minimize is the following:

$$MSE_{critic} = N^{-1} \sum \left(Q_{critic}(s, a) - (r + not\_done * \gamma * Q_{critic_{TARGET}}(s', \pi_{TARGET}(s'))\right)^2$$

Actor

The Actor is a deterministic policy that must choose the action that maximize Q_critic, then the Actor loss function to minimize is given by:

$$Loss_{actor} = -N^{-1} \sum Q_{critic}(s, \pi(s))$$

Soft Target Update

At the end of each training step, the parameters of the target networks are partially updated with the parameters of the current networks using a parameter tau<<1.

$$\sigma_{TARGET} \leftarrow \tau * \sigma + (1 - \tau) * \sigma_{TARGET}$$

Behavior Policy
Since DDPG is an off-policy algorithm, to act in the environment, the agent doesn't use the Actor but a behavior policy Beta built from the Actor adding a mean-zero Gaussian noise that encourages exploration.

$$\beta(s) = \pi(s) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma_\beta)$$

1.3 TD3

Twin delayed deep deterministic policy gradient (TD3) method is an improvement of DDPG algorithm that suffers mainly from two problems:

- Overestimation bias
- High-variance estimates

1.3.1 Addressing overestimation bias

As it happens in Q-Learning with discrete action space, also action-crituc method overestimates the action value at each iteration, although this overestimation may be minimal, if left unchecked, it could grow over many updates, moreover an inaccurate value estimate may led to poor policy updates. In order to reduce this problem, TD3 uses a technique called Clipped Double Q-Learning in which two different critics are used and to compute the target y it takes the minimum between the two estimates.

$$y = r + not\_done * \gamma * \min_{i=1,2} Q_{critic_{TARGET_i}}(s', \pi_{TARGET}(s'))$$

1.3.2 Addressing variance reduction

High variance estimates are a problem because they provide a noisy gradient for the policy update that causes the reduction of learning speed and performance. Since the high variance is proportional to the estimation error, TD3 delays the policy update in order to give time to the critic to reduce error and stabilize itself, moreover to reduce the growth of error, also the target networks update is delayed.

Another cause of variance increase is the use of a deterministic policy that can overfit to narrow peaks in the value estimate. TD3 reduces this problem by adding a clipped noise to the action chosen by the actor in the computation of the target y, this noise corresponds to a regularization term. Then the target y is given by:

$$y = r + not\_done * \gamma * \min_{i=1,2} Q_{critic_{TARGET_i}}(s', \pi_{TARGET}(s') + \epsilon) \quad \epsilon = clip(\mathcal{N}(0,\, \sigma_a), -c, c)$$

1.3.3 TD3 algorithm

Starting from DDPG algorithm and then making the improvements explained above, the overall TD3 algorithm is the following.

Algorithm 1 TD3 Algorithm

Initialize critic networks Q_1, Q_2 with random parameters θ_1, θ_2
Init actor network π with random parameters ϕ
Init target networks $\theta_{TARGET_1} \leftarrow \theta_1$; $\theta_{TARGET_2} \leftarrow \theta_2$; $\phi_{TARGET} \leftarrow \phi$
Init Replay Buffer B
for `t = 1...T` **do**
    //Exploration
    $a \leftarrow \beta(s)$ ▷ $\beta(s) = \pi(s) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma_\beta)$
    Execute action a and store $< s, a, r, s', not\_done >$ into B

    //Training
    Sample mini-batch of N transitions $< s, a, r, s', not\_done >$ from B
    $\tilde{a}' \leftarrow \pi_{TARGET}(s') + \epsilon$ ▷ $\epsilon = clip(\mathcal{N}(0, \sigma_a), -c, c)$
    $y = r + not\_done * \gamma * \min_{i=1,2} Q_{TARGET_i}(s', \tilde{a}')$
    Optimize Q_i with $MSE_i = N^{-1} \sum (y - Q_i(s, a))^2$
    if `t mod d` **then**
        //Training Actor
        Optimize π with $Loss = -N^{-1} \sum Q_1(s, \pi(s))$

        //Soft update target networks
        $\theta_{TARGET_i} \leftarrow \tau * \theta_i + (1 - \tau) * \theta_{TARGET_i}$
        $\phi_{TARGET_i} \leftarrow \tau * \phi_i + (1 - \tau) * \phi_{TARGET_i}$
    end if
end for

1.4 HER

Hindsight Experience Replay (HER) is a technique that allows the learning in an environment with sparse and binary reward, without this method it is usually necessary to use a complicated reward engineering based on the specific environment.

HER can be used with any off-policy method but requires an environment with multiple goals. The key idea behind this technique is to make the agent able to learn from its mistakes, for example, considering an agent that must reach a target position (goal), from an episode in which it didn't reach the goal, it can learn how to reach the final position or another intermediate state.

In a multi goal environment, the policy has as input the observation and the current goal, during the exploration phase the algorithm stores the transition normally, but at the end of the episode it uses HER by adding some artificial transitions building from each real transition with different goals and the reward computed according to the new goals. So another important requirement for the algorithm is the access to the reward function of

the environment. The new goals can be chosen with different manners, the HER's authors find out that the best strategy is 'future' in which for each transition, the algorithm adds k different transitions with as goal a state reached during the episode after the current transition.

The algorithm is the following:

Algorithm 2 HER Algorithm

Initialize off policy algorithm $\mathcal{A}$
Init Replay Buffer B
for `episode=1...M` **do**
  Sample an initial state s_0 and a goal g
  for `t = 1...T` **do**
    //Exploration with behavioral policy from $\mathcal{A}$
    $a_t \leftarrow \beta(s_t|g)$
    Execute action a_t and store $(s_t|g, a_t, r_t, s_{t+1}|g, not\_done)$ into B
  end for
  for `t=1...T` **do** ▷ HER with future strategy
    Choose k states s_j such that $j > t$
    for `j = 1...k` **do**
      $g_j \leftarrow s_j$ ▷ New goal
      Compute new reward r_j
      Store $(s_t|g_j, a_t, r_j, s_{t+1}|g_j)$ into B
    end for
  end for
  for `opt_step = 1...P` **do**
    Sample mini-batch of N transitions $(s|g, a, r, s'|g, not\_done)$ from B
    Perform one step of optimization using $\mathcal{A}$ and the mini-batch
  end for
end for

2. DEVELOPMENT

The project has been developed in three phases, in the first the TD3 implementation has been tested, in the second, HER technique has been used in the most easily robot environment, then in the last phase, the algorithms have been applied on pick and place robot environment.

2.1 TD3 TEST

The code has been developed using Tensorflow2, the parameters used during the train are shown in Table 1.

Parameter name	Reference to Algorithm 1	Value
EXPLORATION_NOISE	σ_β / max_value_action	0.1
BATCH_SIZE	N	256
POLICY_NOISE	σ_α / max_value_action	0.2
NOISE_CLIP	c / max_value_action	0.5

Parameter name	Reference to Algorithm 1	Value
DISCOUNT	γ	0.99
POLICY_FREQ	d	2
TAU	τ	0.005

Table 1: Parameters TD3 implementation

Actor and critic networks have two hidden layers composed of 256 nodes with ReLu activation function. The algorithm has been tested with 'HalfCheetah-v2' OpenAi gym environment and trained for 200.000 steps using a Nvidia GPU GeForce GTX 1050.

Figure 1 shows the improvement of the agent considering the average reward in ten episodes.

The performance achieved is in line with the result obtained by TD3's authors with HalfCheetah-v1 environment.

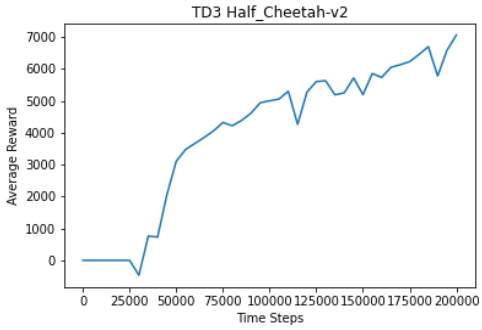


Figure 1: Average Reward over 200.000 steps

2.2 TD3+HER TEST

In order to test the implementation of HER technique with TD3 algorithm, the code has been used with the most easily robot environment 'FetchReach-v1' in which the robotic arm must reach a position in a 3D space with its gripper.

To make possible the learning in this type of environment with neural networks, it is needed to use a normalization technique for observation and goal, to do that, a Gaussian normalizer has been adopted.

The parameters of the distribution, mean μ and standard deviation σ, are estimated in the following way:

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x_n \qquad \hat{\sigma} = \sqrt{\max(\epsilon^2, \frac{1}{N}\sum_{n=1}^{N} x_n^2 - \hat{\mu}^2)}$$

X_n represents an observation or a goal collected during the exploration of the environment, to compute the goal normalizer have also been considered the artificial goals added with HER. The parameters ϵ is fixed to 10^-2 and it necessary to avoid

division by zero during the computation of the normalized quantity.

$$x_{norm} = clip\left(\frac{x - \hat{\mu}}{\hat{\sigma}}, -z, z\right)$$

During the training, the normalized quantities have been clipped between -5 and 5. (z=5)

The agent has been trained for 2 epochs, each epoch is composed of 50 cycles which in turn are composed of 16 episodes. The parameters of TD3 algorithm used are the same as in the previous case, except for the dimension of the mini-batch reduced to 128 samples. In HER method for each step, four additional goals are chosen and at the end of each episode fourty optimization steps are performed (k=4, P=40).

The agent achieved good performance quickly, indeed it reached 96.40% success rate at the end of first epoch as shown in Figure 2

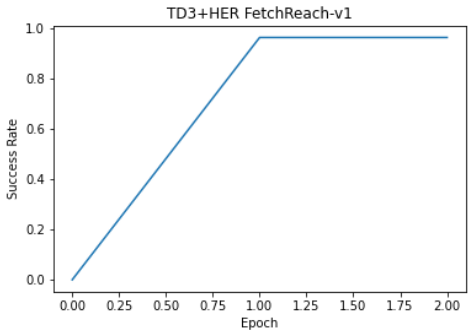
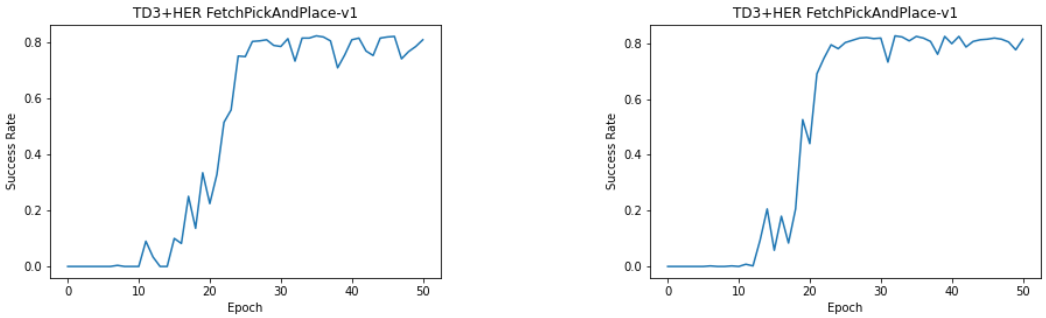


Figure 2: Success Rate over 2 epochs

2.3 TD3+HER PICK AND PLACE

In both experiments above, at the beginning of the train, has been used a trick to improve the exploration at the start time in which for the first S steps the action has been chosen sampling from a uniform random distribution, according to TD3'a authors, S has been set to 25.000. In Figure 3 is shown a comparison between the performance reached by the agent with and without this trick in 'FetchPickAndPlace-v1' environment.



(a) (b)

Figure 3: Comparison between the performance reached without (a) and with (b) random start

It is possible to notice that with random start the agent can learn quickly, indeed considering for example epoch 20, without this trick it obtains about 20% of success rate, instead, with this method, about 50%.

Finally the agent has been trained for 100 epochs with random start and using the same parameters described above, the training took about 24 hours with a Nvidia GPU GeForce GTX 1050. In Figure 4 are shown two training sessions with different random seeds.

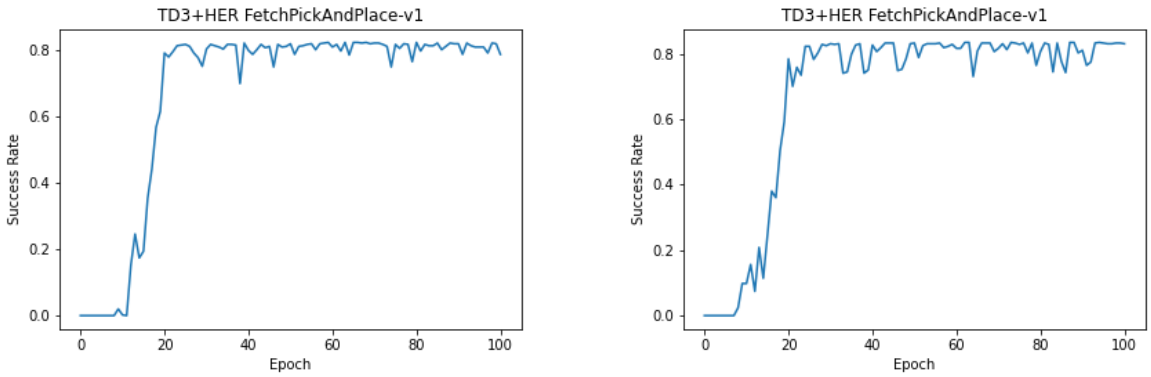


Figure 4: Training with random start over 100 epochs using two different random seeds

As it is shown in the plots, the agent reaches a good performance after about 30 epochs and it remains at about 82% of success rate, continue training the model is necessary to improve the stability of the robot gripper once it reaches the goal position.

3. CONCLUSIONS

In the project has been implemented TD3 algorithm with HER technique to train a robotic arm to perform pick and place task, the success rate reached is about 82% after the training of 100 epochs using a GPU, the performance are less than those obtained by HER's authors with DDPG algorithm. In order to improve the performance it is necessary to change the code to perform the computation over several CPU threads as done in the paper.

REFERENCES

[1] OpenAI **Hindsight Experience Replay**

[2] OpenAI **Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research**

[3] Scott Fujimoto, Herke van Hoof, David Meger **Addressing Function Approximation Error in Actor-Critic Methods**

[4] OpenAI **Twin Delayed DDPG — Spinning Up documentation:**
https://spinningup.openai.com/en/latest/algorithms/td3.html

[4] OpenAI **DDPG — Spinning Up documentation:**
https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[5] Andrea Lonza **Reinforcement Learning Algorithms with Python**