Report
Reinforcement Learning Final project

Fabrizio Casadei

Matriculation number: 1952529

SAPIENZA
UNIVERSITÀ DI ROMA

February 20, 2021

Chapter 1

Overview

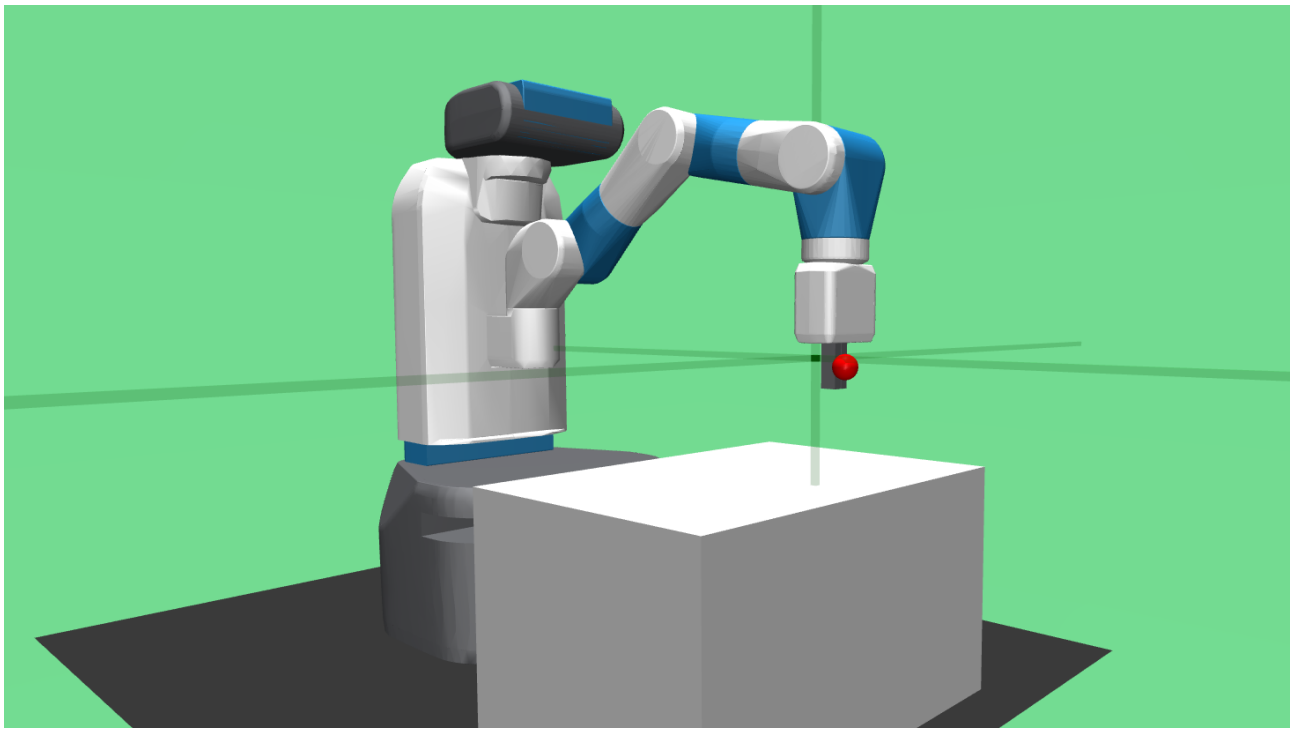
The aim of this project is to solve a Reinforcement learning problem, using one of the environments provided by *Open AI*.
The assigned project, consists in implementing the *Soft Actor Critic* algorithm, with an additional technique called *Hindsight Experience Replay*.
All the code must be implemented using the *Pytorch* Framework and *Python* as the programming language.
Very briefly, we want to introduce the aim of *Reinforcement learning*.
The goal of RL is to learn a behaviour for an *agent* based on trajectories composed of states, actions and rewards elements. Hence, we want to map each state of our environment to an action for maximizing the reward.

1.1 The environment

The *Open AI* environment assigned for this project, is called "FetchReach-v1", which is part of the robotics environment set.
The goal is to control the end effector of the robotic arm, to reach a specific, randomly chosen, position in the 3D space, as quickly as possible. Therefore, the goal is 3-dimensional and describe the target position. Rewards are sparse and binary, we get a reward equal to 0 if the end-effector is the goal position, otherwise -1.
The actions are 4-dimensional, 3 values express the end-effector position, while the last one control the opening and closing of the gripper (useless in this specific case).
The observations include a lot of more data, like end-effector linear velocity as well as the position, linear velocity of the robot's gripper, so on and so forth.

This problem is an example of *Multi-goal RL*, in which we are interested in learning agents to achieve multiple goals. We train *policies* and *value functions* (*state-value functions* & *action-value function*), that takes as input both states and goals. In this case states, or better, observations are described by these 3 following properties:

- **observation**, the actual observation of the environment, For example, robot state and position of objects.
- **desired_goal**, the goal to achieve
- **achieved_goal**, the goal that the agent has currently achieved, hence, the actual state.



The environment "FetchReach-v1"

Chapter 2

Background

2.1 Soft Actor Critic

The *Soft Actor critic* is a state of the art algorithm used in *Reinforcement Learning* which is based on the *off-policy* maximum entropy deep reinforcement learning (usage of neural networks) approach.
We quickly recall the concept of *off policy* algorithm: in this case we utilize 2 types of policies, *the behaviour policy*, for determining the agent action given a certain state, and *the target policy*, a policy used from the agent to learn from rewards received for its action.
In an off-policy algorithm is possible to store and append past experience, in a buffer, called either **Replay buffer** or **Experience Replay**. This buffer is used to retrieve older samples provided through older policies and use them to update the actual policy.
SAC as mentioned, center its operations around the concept of **Entropy**, but what does it mean? What is the role of the Entropy?
The *Entropy* is a quantity that corresponds to the randomness, referring to a variable.
SAC exploits the *entropy regularization* in this way, the policy is trained to maximize the trade-off between expected return and the entropy, this can be seen as the concept of handling exploration and exploitation, increasing *entropy* means having more exploration, which can improve the further exploitation.
Maximum entropy formulations provide a substantial improvement in exploration and robustness of the results, policies are robust towards models and error estimations.
The policy tries to achieve the goal kipping the entropy high, which means, whether there are 2 possibilities to achieve the goal, then the policy assigns equal probability to those, obviously, this happens only if the 2 actions are equal good, otherwise, the probabilities would be different, according to the action worth.
SAC keeps a high level of exploration, letting all the possibilities open, also due to the stochastic actor, which removes the deterministic choice from the policy.

To sum up we can highlight these 3 main components for the *Soft Actor Critic* algorithm:

- **actor-critic architecture**,which separates policy and value functions. Value function and Q function are not directly estimated, but they are evaluated using the neural networks by sampling information from the environment.
  The utilization of Neural networks works, because they are universal function approximators, through the re-parametrization they become more accurate after each update.
- **Retrieval of past data**, using the *Experience replay*, the off-policy formulation reuse the previously collected data.
- **Entropy maximization**, to enable stability and exploration.

this algorithm, hence, define e new goal for the agent, not only we want to maximize the cumulative rewards but also we want to maximize the entropy simultaneously, therefore, we have to define e different *objective function*, shown below.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot \,|\mathbf{s}_t)) \right]$$

"The new objective function"

The summation this time is done taking into account not only the reward component, but also considering the entropy of the policy, regularized by the temperature parameter (α).

In the end, to conclude this brief introduction to the algorithm, it's important to highlight *SAC*'s problems, it is characterized by having a brittle convergence and a very high sample complexity (the first is given due to the utilize of neural networks), in high dimensional states and observations, it's hard finding the convergence respect to the hyperparameters.

A possible pseudo-algorithm of the *SAC*, is the following:

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  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 j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \Big(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta \left(\tilde{a}_\theta(s)| \, s\right) \Big),$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
      Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$

    end for
  end if
until convergence

"possible implementation"

We will discuss another possibility of development for the target function in the Third chapter, beyond that, this is the schema used to project the algorithm.

2.2 Hindsight Experience Replay

Hindsight Experience Replay, or more simply, *HER*, is a particular technique used in *off-policy* and *Multi-Goal* RL algorithm, to learn from the outcomes that you get from your action, virtualizing a specific goal, which is the one you have reached after a certain number of steps in the episode.
this "virtual" goal, is exchanged with the effective goal for the episode, and then, the corresponding transition, is stored with this new goal.
The basic intuition is thinking about when we fail after doing a specific action, the desired goal hasn't been satisfied, but if we think about the reached goal, as a possible requested goal for the environment, we can use these data as a positive trajectory to store in the *experience buffer*.
In this manner, it allows for increasing the speed of convergence of the learning process.
There are many ways to apply this technique on our project, and a general pseudocode, is the following:

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 $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$. ▷ 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

HER pseudo-code

Therefore, after experiencing some episode, $S_0, S_1, ..., S_T$ we store in the *experience buffer* not only with the original goal of the episode, but with a subset of other goals (could be also only one).
More implementation aspects are described in the next chapter.

Chapter 3

Implementation

In this chapter, we are going to point out implementations about core parts of the project, which are been introduced theoretically in the previous section. Before doing this, it's important to highlight some implementation choice, some of these, not strictly related to *SAC* & *HER*:

- **The reshape**, as expressed in the first chapter, the observation are characterized from several values. The choice taken has been the reshaping of observation, extracting the *achieved goal* and the *desired goal*. All other information concerning the end-effector motion are been ignored. This because better performance has been recorded, with fewer input values to handle.

- **The main loop**, manage the environment execution, for each step, and for each episode.
  Define several attributes and functions for the final evaluation of results, end through several boolean flags, add different features to the execution, like the *Hindsight Experience Replay*. Here the value of transitions are handled and used in the *Experience replay* and given to the agent, for learning and making decisions about the next action.

- **Actor network**, has been modelled with a very simple structure, made up of 2 hidden layers of 256 units and using as activation functions the *ReLu*. the output layer has units with an equal number of actions for the environment, and uses the *linear* activation function.
  The policy is treated as a *Gaussian* with *mean* and *standard deviation* given by the neural network.
  From the actor, action and entropy related to the action are provided. The first is randomly sampled from the Gaussian probability distribution, then is bounded using the *tanh* function and the max value for action (the high property). while the entropy, is computed as the logarithmic probability, associated to that action, as declared in the following formula:

$$H(P) = \underset{x \sim P}{\mathrm{E}} \left[-\log P(x) \right].$$

  Then is adjusted applying the *Enforcing Action Bounds* formula, to limit the action distribution.

- **Critic networks**, are composed by 4 hidden layers, each one with 256 units and *ReLu* as the activation functions. The output layer is composed of a single unit with *linear* activation function.
- **Value networks**, value and target value networks, have a very similar structure relative to the critic, but are made up of only 2 hidden layers, (also the input changes, in this case, it's characterized by the observation values, while in the critic network are added the action values).
- **Optimizers**, For all the networks, has been chosen as optimizer the *Adam*, which interact directly with parameters.

3.1 SAC algorithm

In the Background chapter, we have highlighted, through a pseudocode, the main steps to realize the *Soft actor critic* algorithm. In the implemented project, we estimate the target value in a different manner.
starting step by step, let's see how has been managed the whole execution flow.
We start initializing, with arbitrarily chosen parameters, five neural networks characterized in this way, one for the policy, two for the critic, one for the value and one for the target value.
Our algorithm makes use of two Q-functions to mitigate positive bias in the policy improvement step.
in the main loop, we observe the actual state and then (after a certain number of step, otherwise randomly) execute the action suggested from the policy. all the data which come from the execution, like the new state, the reward and the termination flag, is registered together with the actual state and the action performed, in the *Experience replay*.
Whether it's time to update the parameters of networks (after a N predefined number of step, update each M step), we extract data from the buffer and call the "learn" function of the *SAC agent*.
here, are estimated three kinds of loss, the *policy loss*, the Q *loss* and the V *loss*.
These loss values are used to train and update the parameters of actor, critic and value networks. Differently the target value network will be updated manually using the *Polyak* hyperparameter, that describe in which percentage the parameters will be changed, respect to the ones of the value network.

the *Q loss* it's estimate through the calculus of the target value, that is given directly from the target network. Another possibility, as announced, is to compute it, in the following way:

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t | \mathbf{s}_t) \right]$$

In that case, we don't need to define a value network. Our choice has been, due to stability reason, the opposite, that's the definition of an appositive network and we have used the above formula as target for the update of the V function.
After having computed, thanks to the *bellman backup*, the Q value according to the target, we estimate the two losses (two because we have double q functions) using the *mean squared error*.
At last, we take as final Q loss value the sum of the two.

The *V loss*, is computed taking the action and the entropy from the actor and using that action to compute the two Q values from critics. the target V value becomes the minimum of these two, and it's adjusted making it "soft", that is by subtracting the entropy (multiplied by the α). Finally, the *mean squared error* is evaluated using the estimated value function from the V network, and the target.

The *policy loss*, it's performed, subtracting the α entropy value to the minimum of the double Q functions, and taking the mean.

for each parameters update, first, it's set the gradient to zero, then the corresponding loss it's calculated and the backward operation it's performed, in the end, the optimization algorithm is used.

3.2 HER algorithm

In the previous chapter, we have introduced the concept behind *Hindsight Experience Replay* technique, and how said, there are different ways of implementing this. The most important difference is in the *Strategy* used to sampling goals during the episode. First of all we introduce the parameters used for handling the *HER* technique:

- **HER_enabled**, a simple boolean flag, used to anable or disable *HER* in the algorithm
- **HER_StartEpisode**, the episode number used as reference for starting *HER*
- **HER_GoalsStep_n**, number of steps to sample a new virtualized goal, for every episode.

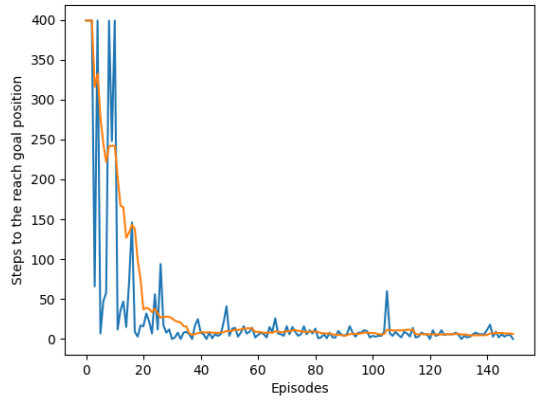
The most simple way to implement a strategy, is using as a new goal, the last achieved goal of the final step in the episode.
Here, we do something, slightly different, we haven't stored a single goal for each episode but G goals, with G given by the parameter
HER_GoalsStep_n and the maximum number of steps in the episode.
for each step in an episode, we sample the corresponding transition, and every
HER_GoalsStep_n steps, a new virtualized goal, which is equal to the achieved goal of the new observation.
we used these two data, after having completed the episode. In fact, we have added a new loop, for the stored transitions, in which is computed the new virtualized reward and its termination flag. Then states are re-shaped and finally saved the new *trajectory* in the *experience buffer*.
in the pseudo-code presented in the previous chapter, the learning or updating step is done, with another loop after this step, while in the presented code for the project, updates are handled in the loop of the steps, just before this implementation of *HER*.

Chapter 4

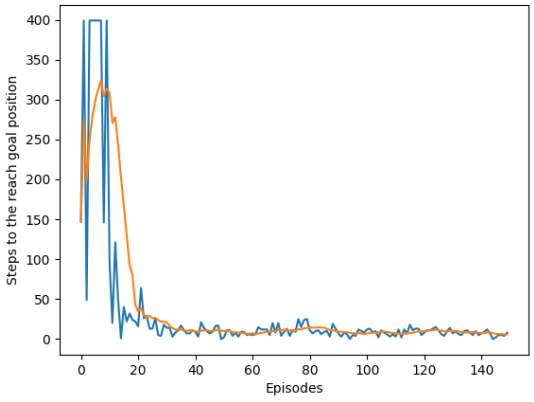
Conclusion

In the end, to conclude this report, it's important to show the results achieved.
It's not always easy to achieve convergence using RL algorithms, above all in a continuous environment, exactly for this reason we could be very satisfied with the outcomes registered. The Soft Actor Critic, a state of the art RL algorithm, has shown all its performative potential, reaching good results with a relatively fast execution.

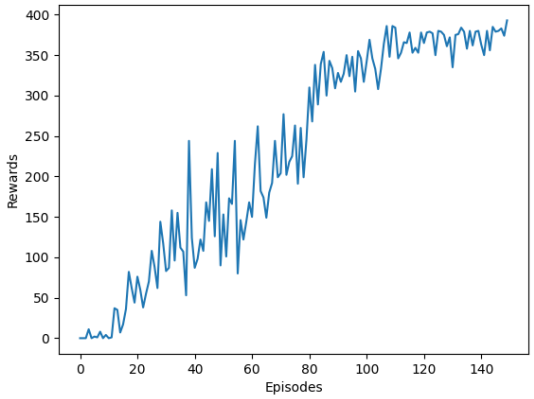
Follows, Plots obtained using only SAC and others with the addition of HER, just to compare the two executions.
First, we do a comparison about the speed of the two variants, the blue lines represent the step number to reach the right position in the 3d space, while the orange lines, an average of these number of steps, obtained considering the last 5 episodes.
Straight away, we evince plots for rewards and average rewards evaluated in both cases.
For all these plots, we have tested the environment for 150 episodes, each one, composed of 400 steps.



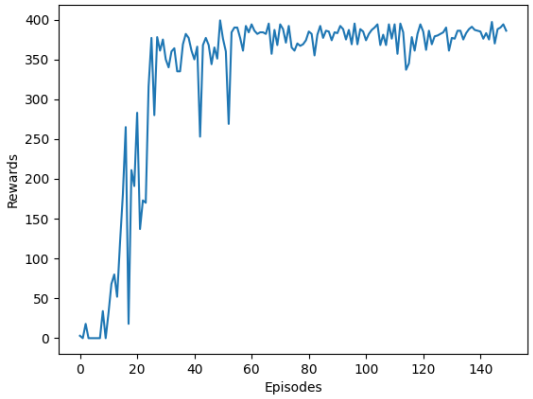
SAC without HER,
step to reach the goal position



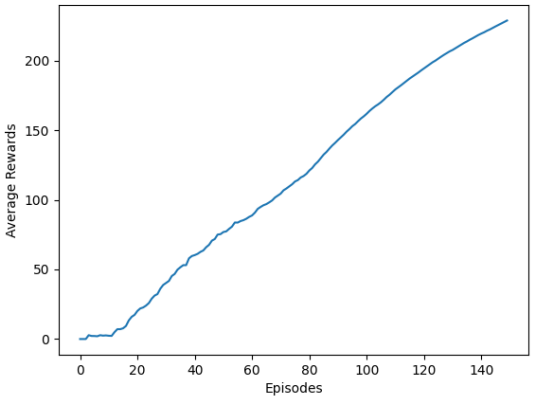
SAC using HER,
step to reach the goal position



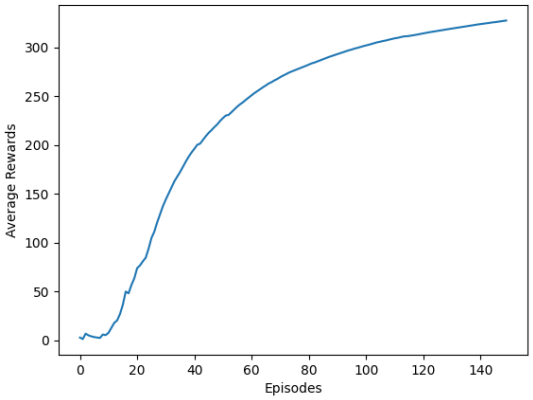
SAC without HER, rewards



SAC using HER,rewards



SAC without HER, average rewards



SAC using HER, average rewards