Robot Learning Homework 4

Carena Simone, s309521
s309521@studenti.polito.it

1 Introduction

The purpose of this assignment is to develop implement and test a simple REINFORCE algorithm. In particular the system for which it's implemented is the *cartpole* system, but is here is considered a continuous action space instead of a discrete one. In fact the force applied to the system is now a single real number.

After that, more advanced *Actor Critic* methods are explored. In particular the *Proximal Policy Optimization* (*PPO*) and *Soft Actor Critic* (*SAC*) are tested and compared.

2 Policy-Gradient

REINFORCE is a policy gradient algorithm. This methods learn a parameterized policy $\pi(a|s, \boldsymbol{\theta})$, whose parameters $\boldsymbol{\theta}$ are updated in order to maximize a scalar objective function $J(\boldsymbol{\theta})$. To achieve such result, the parameters of the policy are updated in the direction of maximum increase with respect to the objective function

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}_t)$$

where, thanks to the *Policy Gradient Theorem*, we have that $\nabla J(\boldsymbol{\theta}) = \mathbb{E}\left[\nabla \log \pi(a|s, \boldsymbol{\theta}_t) Q_\pi(s, a)\right]$, which is equivalent to $\mathbb{E}\left[\nabla \log \pi(a_t|s_t, \boldsymbol{\theta}_t) G_t\right]$, having that G_t is an unbiased sample of $Q(s, a)$. This policy gradient method can be generalized to include a *baseline* term $b(s)$. The baseline can be any function, with the constraint that it must not depend on the action a. This corrective term leaves the expected value of the update unchanged, but it can have an effect on its variance, it may in particular reduce it. The update rule, considering also the baseline results in

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t - b(s_t)\right) \nabla \log \pi(a_t|s_t, \boldsymbol{\theta}_t)$$

A comparison of the result obtained for the cartpole system, with and without a baseline, is presented in the following sections.

2.1 REINFORCE with and without Baseline

Without using a baseline, the results obtained during the training process are reported in (1), while the results obtained using a constant baseline $b(s) = 20$ are reported in (2). Lastly, a final version is proposed, where instead of the return G_t for the update, it is considered the normalized return $\tilde{G}_t$ such that $\mu(\tilde{G}_t) = 0$ and $\sigma(\tilde{G}_t) = 1$. This normalization is obtained as $\tilde{G}_t = \frac{G_t - \mu(G_t)}{\sigma(G_t)}$. The results obtained using this method are reported in (3).

A baseline function is added to the policy gradient method to try to reduce the variance of the gradient estimates. This reduction in variance also helps to speed up the training process. The baseline function can be chosen arbitrarily, under the only constraint that it must not depend on the actions a. A reasonable choice for the baseline is an estimate of the state-value function $\hat{v}(s, \mathbf{w})$. This helps to set the baseline based on the state we are currently in.

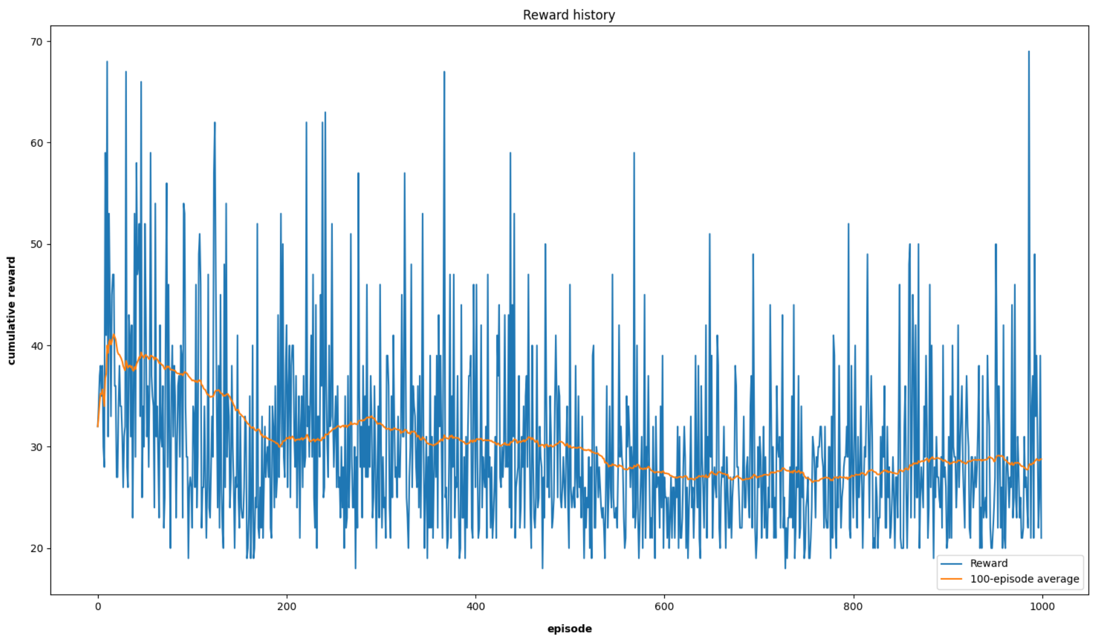

Figure 1: REINFORCE without baseline

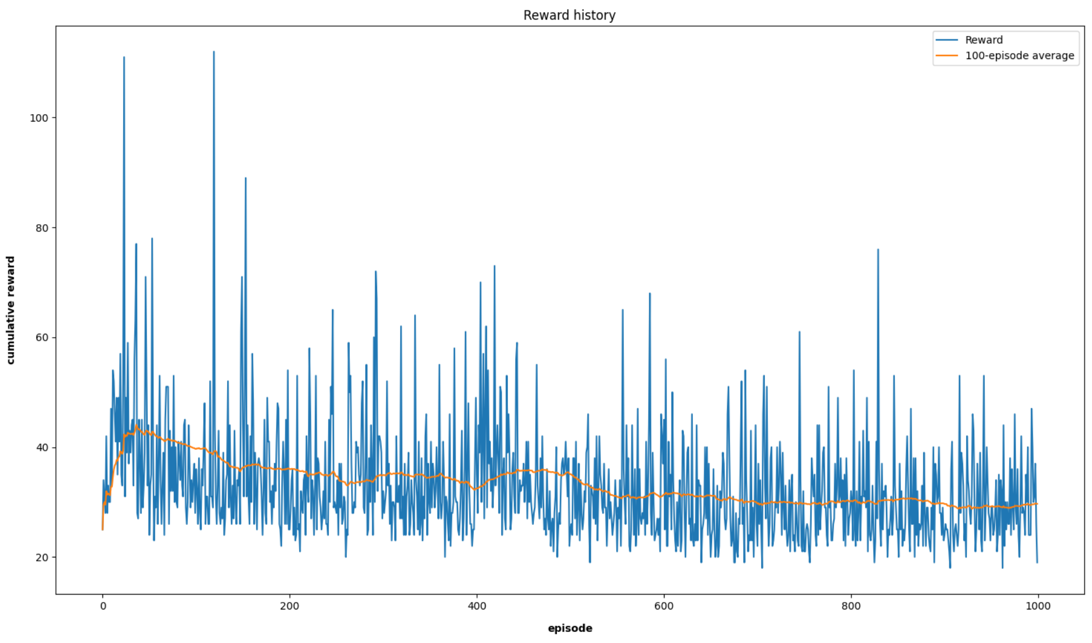

Figure 2: REINFORCE with a constant baseline $b(s) = 20$

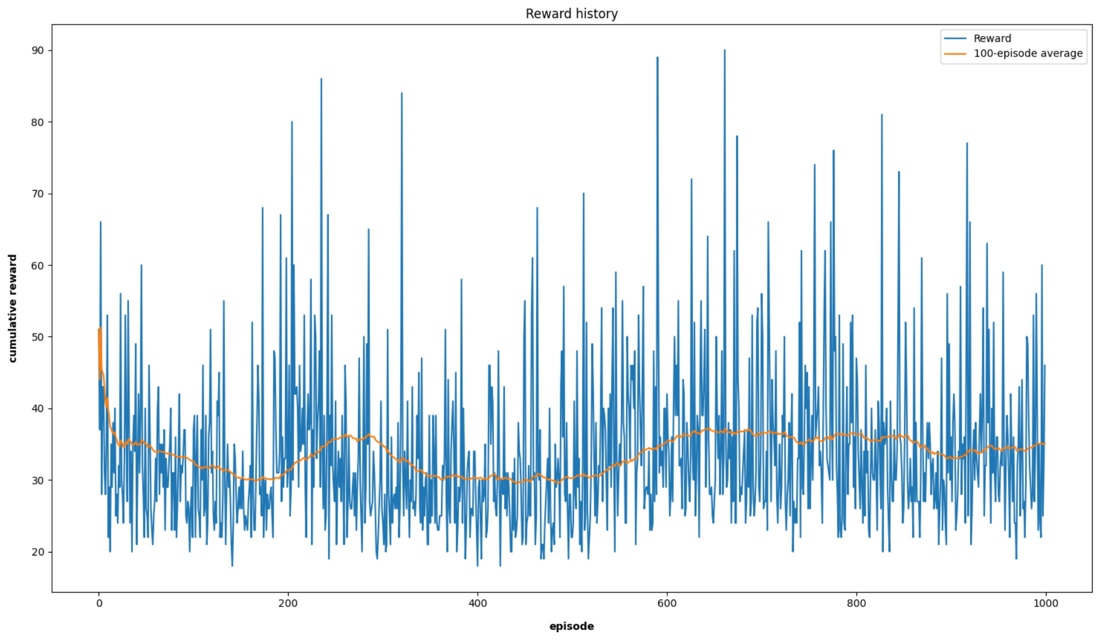

Figure 3: REINFORCE with normalized return

2.2 Real World Scenarios

When dealing with real physical systems a variety of problems may arise during the training phase, due to the limited interactions the system can have with the surrounding environment and the structural limitation of the system itself. Having a continuous unbounded action space may lead to complications and problems to the component constituting the system. The agent could, in fact, generate an action that would result in a command effort exceeding the maximum threshold for the actuators. This stress on the system may result in damaging some of the component, since the the actuators might have to work at saturation for a prolonged amount of time. Another issue comes from having as reward function `+1` for each instant the system manages to stay alive. This encourages the agent to learn a good behavior by itself, without human directions. This comes at the cost of having to try and fail possibly a great amount of times. This is a feasible in simulation, though it may take a long time, but becomes impossible for a real physical system. The agent may (and probably will) attempt action leading to its impairment, thus breaking it.

A way to avoid the problem deriving from the effects of the unbounded action space would be to give a penalization term to the agent in case it takes undesirable actions, such as saturating the actuators for too long. The penalty term in this case could be proportional to the total energy generated by the actuators. Even though this could help managing the real world system after the training has happened, the consideration regarding the agent incurring in dangerous actions in the learning process remains. If the reward is shaped to make the agent more careful it may learn too slowly or avoid possible optima actions and get stuck in a sub-optimal policy. On the other hand, if it is free to try any possible actions it may break itself in the process. Training on a real physical system remains problematic, and the best course of actions would be to train the agent in simulation.

2.3 Discrete Action Spaces

Policy gradient methods can also be applied to discrete action spaces. In case of a discrete action space a common parametrization for the policy is obtained by weighting actions using a linear combination of features $\boldsymbol{\phi}^{\mathrm{T}}(s, a)\boldsymbol{\theta}$. The actions with the highest weight in each state are assigned the highest probability using a softmax function

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{\boldsymbol{\phi}^{\mathrm{T}}(s,a)\boldsymbol{\theta}}}{\sum_k e^{\boldsymbol{\phi}^{\mathrm{T}}(s,a_k)\boldsymbol{\theta}}}$$

which in turn gives

$$\nabla\pi(a|s,\boldsymbol{\theta}) = \phi(s,a) - \mathbb{E}_\pi\left[\phi(s,)\right]$$

3 Actor-Critic Methods

Actor-Critic is a Reinforcement Learning algorithm that combines value-based and policy-based methods. The method is somewhat similar to REINFORCE with baseline $\hat{\nu}(s_t, \mathbf{w})$, but instead of the full return G_t, the n-step return $\sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n \hat{\nu}(s_{t+n}, \mathbf{w})$ is used. The resulting update step for the policy parameter is thus

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(\sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n \hat{\nu}(s_{t+n}, \mathbf{w}) - \hat{\nu}(s_t, \mathbf{w}) \right) \nabla \log \pi(a_t|s_t, \boldsymbol{\theta}_t)$$

Two popular actor-critic algorithms are *Proximal Policy Optimization* (*PPO*) and *Soft Actor Critic* (*SAC*). Testing these two methods on the cartpole system for 200000 episodes gives the results in (4) and (5) respectively. The training took approximately 1.47 min for PPO and 21.28 min for SAC.

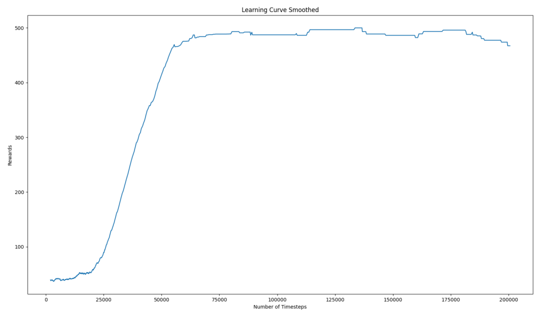

Figure 4: PPO

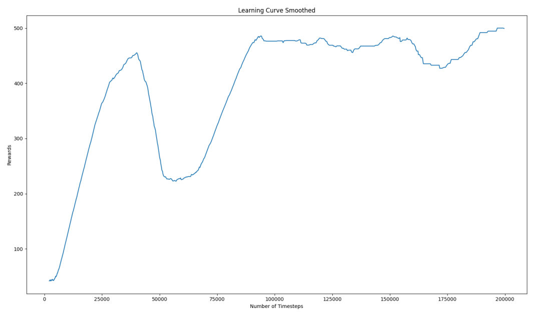

Figure 5: SAC

As it is possible to see from the plots, the PPO methods is slower to reach an higher reward, but it is also more stable in the process. The SAC algorithm yields an higher reward faster, but its tendency to explore makes it deviate from the optimal behavior to try some other actions. This behavior is due to the fact that SAC incorporates an entropy term in the objective function, which incentives the agent to explore more.