# Robot Learning Homework 4

**Matteo Zulian s310384**
Automation and Intelligent Cyber-Physical Systems
Politecnico di Torino

This exercise asks to focus on policy gradient reinforcement learning algorithms, specifically implementing the naïve REINFORCE algorithm and then exploring more advanced Actor Critic methods like Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC), also leveraging on the use of well-known libraries such as Stable-Baselines3. These will be used on a Cartpole environment with continuous action space.

## 1 Introduction to Policy Gradient methods

The policy gradient methods are particularly useful for problems with continuous action spaces. They learn a parameterized policy $\pi(a|s, \theta)$ that can select actions without consulting a value function. A value function may still be used to learn the policy parameter (Chapter 5), but is not required for action selection.

Policy Gradient methods involve learning the policy parameter based on the gradient of some scalar performance measure $J(\theta)$ with respect to the policy parameter. These methods seek to maximize performance, so their updates approximate gradient ascent in $J$:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta) \tag{1}$$

where $\nabla J(\theta)$ is a stochastic estimate whose expectation approximates the gradient of the performance measure with respect to its argument $\theta$.

## 2 REINFORCE Algorithm

### 2.1 Without baseline

REINFORCE Algorithm is a Policy Gradient method where the update of the parameter $\theta$ is given by:

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{2}$$

When the agent takes an action $A_t$, the policy $\pi_\theta$ is updated proportionally to the direction in parameter space that most increases the probability of repeating the action on future visits to state $S_t$, this because if action $A_t$ was a good action then the agent wants to choose it more often so it increases $\theta$ in that direction. This solution would be susceptible to initial conditions if it didn't divide by the probability of taking that action, otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return. It makes sense to make the update proportional to the discounted return $\gamma^t G_t$, so that a positive returns increases the probability of an action and vice versa for a negative one.

### 2.2 With baseline

REINFORCE algorithm can be generalized to include a comparison of the action value to an arbitrary baseline $b(s)$:

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

$$\theta \leftarrow \theta + \alpha(G_t - b(s)) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \tag{3}$$

$b(s)$ can be any function, even a random variable, as long as it does not vary with respect to the action otherwise it could introduce bias into the gradient estimates.

One natural choice for the baseline is an estimate of the state value $v(s)$, in this way the update is proportional to $(G_t - v(s))$ which intuitively says that if the the return is greater than the expected return value $(G_t > v(s))$ then the probability of taking action $a$ has to be increased, and vice versa when $(G_t < v(s))$

In general, the baseline leaves the expected value of the update unchanged, but it can have a large effect on its variance

### 2.3 Results

Three methods where compared in this exercise: one REINFORCE algorithm without baseline, one with constant baseline $b(s) = 20$ and one with discounted rewards normalized to zero mean and unit variance.

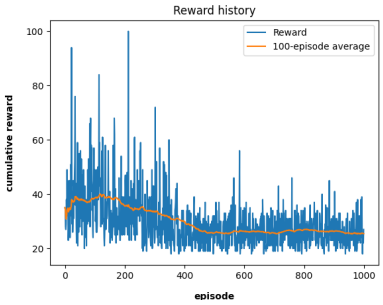



Figure 1: REINFORCE without baseline
Average test reward: 25.21 episode length: 25.21

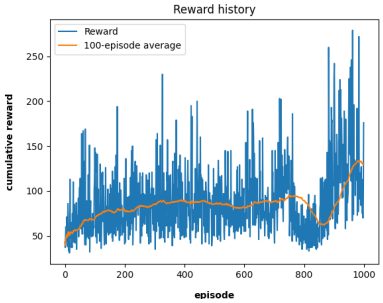



Figure 2: REINFORCE with baseline
Average test reward: 205.95 episode length: 205.95

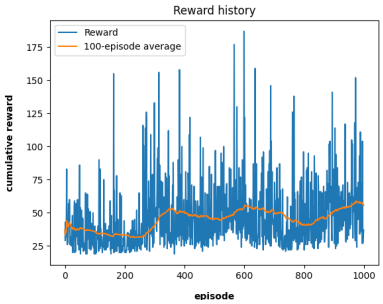



Figure 3: REINFORCE with normalized rewards
Average test reward: 49.46 episode length: 49.46

In reinforcement learning is very important to choose the right rewards to give to the agent, for example in the cartpole system used in this exercise the reward is either 0 or 1. This means that, every time an action is performed and the pole is up, the agent is given a positive reward that will increase the probability of taking that action, otherwise it stays the same. A solution like this impose a non-decreasing policy, that could slow learning or worsen the performance. A baseline of 20 improves this solution because some of the returns $G_t$ become negative. Using a return normalized as $G_t \sim \mathcal{N}(0,1)$ means that the range of values is $[-1,+1]$. This also improves the precision because of the negative values of $G_t$, but in module they are smaller than the values obtained with $G_t - b(s)$ so the learning is slower than the one with baseline.

## 3 Real-world control problems

A unbounded continuous action spaces can make exploration challenging. The model might struggle to explore the entire action space effectively, leading to suboptimal or unsafe behavior. For example in an electrical circuit, an unbounded range of Voltages as action space could be harmful to the components of the physical system or it could saturate to its maximum value. These reasons makes the training slow and unstable.

One solution might be consider incorporating **soft constraints** on the actions, penalizing actions that deviate too far from a reasonable range rather than outright restricting them. Also it could be wise to gradually expose the model to the entire action space by using **curriculum learning**, starting with simpler tasks or constrained action spaces and progressively increasing the complexity. As in many cases of reinforcement learning, the solution could be to properly shape the reward function in order to guide the agent during the training phase to avoid issues like this.

## 4 Discrete action spaces

While Policy Gradient methods are often associated with continuous action spaces, they are quite flexible and can be adapted to handle discrete action spaces as well. If the action space is discrete and not too large, then a natural and common kind of parameterization is to form **parameterized numerical preferences** $h(s,a,\theta)$ for each state–action pair. The actions with the highest preferences in each state are given the highest probabilities of being selected, for example, according to an exponential soft-max distribution proportional to $h(s,a,\theta)$ where $\theta$ can be tuned by a $ANN$ . One advantage of parameterizing policies according to the soft-max in action preferences is that the approximate policy can approach a deterministic policy. When the training starts all actions have the same preferences hence the same probability of being chosen, this means that in the beginning there is a full exploration. Over time these preferences change, ensuring a full exploitation during the last episodes of training. Parameterized numerical preferences are better than $\varepsilon - greedy$ methods where there is always an $\varepsilon$ probability of selecting a random action.

# 5 Actor-Critic Methods

Actor-Critic methods are a class of reinforcement learning algorithms that combine elements of both value-based methods (critics) and policy-based methods (actors).

The actor is responsible for selecting actions based on the current policy. It represents the policy function $\pi(a|s, \theta)$, which maps states to a probability distribution over actions. The actor is often implemented using a parameterized function, such as a neural network, with the parameters denoted as $\theta$

The critic evaluates the actions chosen by the actor. It estimates the expected cumulative reward or value of being in a particular state and following the current policy. The critic's goal is to provide feedback to the actor, guiding it toward actions that lead to higher expected rewards. The critic is also implemented using a parameterized function $v(s|w)$, with parameters denoted as $w$

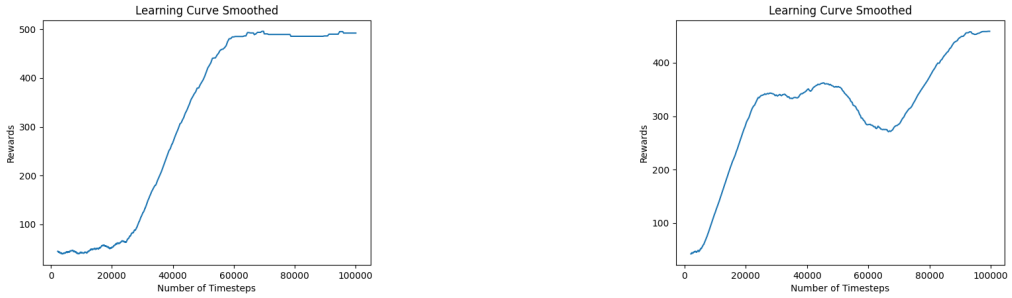
Actor-Critic methods often exhibit reduced variance in comparison to pure policy gradient methods. This can lead to more stable training.

## 5.1 Results

In this exercise two algorithm were analized; Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

PPO is an On-policy algorithm known for its simplicity and ease of implementation and typically requires fewer hyperparameters to tune compared to some other algorithms.

SAC is an off-policy algorithm, meaning it can learn from past experiences more efficiently, making it sample-efficient but increasing its computational demand.




(a) PPO

(b) SAC

Figure 4

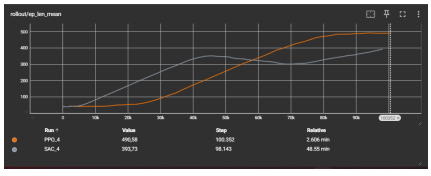


Figure 5: PPO and SAC

As shown in figure 5, in the beginning SAC is faster but after 50000 timesteps PPO catches up and yields a better result. If given enough time, SAC would yield a better mean return than PPO.

Compared to REINFORCE algorithm, PPO and SAC are more complex solutions, but they can be preferred due to their improved stability and sample efficiency.