Robot Learning Exercise 4

December 14th, 2023

TAs: Gabriele Tiboni (*gabriele.tiboni@polito.it*), Andrea Protopapa (*andrea.protopapa@polito.it*), Elena Di Felice (*s303499@studenti.polito.it*)

1 Introduction

This exercise will ask the student 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.

To proceed with the assignment, clone the starting code from the repository at the dedicated webpage and follow the steps listed below in this document.

Your submission should consist of (1) a **PDF report** containing **a discussion of the results and answers to the questions** asked in these instructions, and (2) the **code** used for the exercise.

The **deadline to submit** the report and code through the Github Classroom is **December 28th, at 23:59.**

2 Preliminaries

This assignment asks for the implementation of different policy gradient algorithms. A good preliminary knowledge of the theoretical aspects behind policy gradient and actor critic methods is highly suggested before dealing with the assignment.
In particular, make sure you revise the course slides of Lecture 9, and Chapter 13 of [1].

3 REINFORCE

REINFORCE is a policy optimization algorithm that directly adjusts the parameters of a stochastic policy to maximize expected cumulative rewards and it operates through gradient ascent as optimization algorithm, iteratively searching for optimal parameters that maximize the objective function.

In this section, we will implement this policy gradient algorithm for the Cartpole environment comparing it with and without a constant baseline. We will use a modified version of the CartPole environment implemented in `cp_cont.py` —the standard CartPole uses discrete actions (applying force of either -10 or 10), while in this version it takes actions in the form of a single real number, which represents an arbitrary force value.

3.1 Policy gradient with and without baseline

Task 1 Implement policy gradient for the CartPole environment with continuous action space. Use `agent.py` for implementing the reinforcement learning algorithm itself (for example, the agent and policy classes). These classes are used to implement the main training loop in the `cartpole.py` file, similarly to how it was done in Exercise 1.

Use constant variance $\sigma^2 = 5$ for the output action distribution throughout the training.

Implement the following algorithms:

(a) basic REINFORCE without baseline,

(b) REINFORCE with a constant baseline $b = 20$,

(c) REINFORCE with discounted rewards normalized to zero mean and unit variance,

and compare their results.

Hint: *The* `agent.py` *file from the first exercise session contains some useful code, which you can use as a reference.*

Hint: *Your policy should output a probability distribution over actions. A good (and easy) choice would be to use a normal distribution (*`from torch.distributions import Normal`*). Log-probabilities can be calculated using the* `log_prob` *function of the distribution.*

Question 1.1 How would you choose a good value for the baseline? **Justify your answer.**

Question 1.2 How does the baseline affect the training, and **why?**

3.2 Real-world control problems

Question 2.1 What could go wrong when a model with an unbounded continuous action space and a reward function like the one used here (+1 for survival) were to be used with a physical system?

Question 2.2 How could the problems appearing in Question 2.1 be mitigated without putting a hard limit on the actions? **Explain your answer.**

3.3 Discrete action spaces

Question 3 Can policy gradient methods be used with discrete action spaces? Why/why not? Which steps of the algorithm would be problematic to perform, if any? **Explain your answer.**

4 Actor-Critic Methods

Actor-Critic methods combine the advantages of value-based and policy-based reinforcement learning. In this section, we will explore two popular Actor-Critic algorithms: PPO (Proximal Policy Optimization) [2] and SAC (Soft Actor-Critic) [3].
Stable-Baselines 3 is a state-of-the-art RL library offering efficient and stable implementations of various algorithms. Specifically, it provides robust implementations for PPO and SAC, making it an ideal choice for experimentation due to its ease of use and reliability.

Task 4 Following the official documentation of Stable-Baselines 3, implement inside `cartpole_sb3.py` the code for training and testing a policy on the continuous CartPole environment using either PPO or SAC.

Question 4 Analyze the performance of the trained policies in terms of reward and time consumption. Additionally, compare the results with the REINFORCE algorithm you have previously obtained, highlighting any notable differences in terms of learning stability and convergence speed. **Motivate with some hypotheses** all the results obtained.

Hint: *For the evaluation phase, refer to the "Evaluation Helper" in the documentation.*

Hint: *Feel free to adjust any hyperparameter such as learning rate or others as presented in the documentation if needed.*

Hint: *You can use callbacks for additional functionality, i.e., saving checkpoints, implementing early stopping, or integrating with TensorBoard or WandB for visualization.*

References

[1] "Reinforcement Learning: An introduction (Second Edition)" by Richard S. Sutton and Andrew G. Barto, PDF
[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.*
[3] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.*