Reinforcement Learning 2022
Practical Assignment 3: Policy-based RL

1 Introduction

In the previous assignment, you worked with deep Q-learning, which aims to learn the values of actions in various states. Such a method falls under the umbrella of *value-based* reinforcement learning. In this assignment, in contrast, we are going to investigate the *policy-based* approach to reinforcement learning. In this approach, the policy is optimized directly (i.e., instead of deriving the policy from a representation of the value function).

You will be using the Catch[1] environment shown below. In this task, you need to control a paddle (the yellow block on the bottom) to catch balls (the white blocks in the air) dropping from the top. You can customize the environment in its initialization parameters, making it easier or more difficult by changing configurations such as number of rows, number of columns, speed with which balls drop, the type of observation, etc. Please make sure to **CAREFULLY** read the code and all the comments to understand how the environment works.

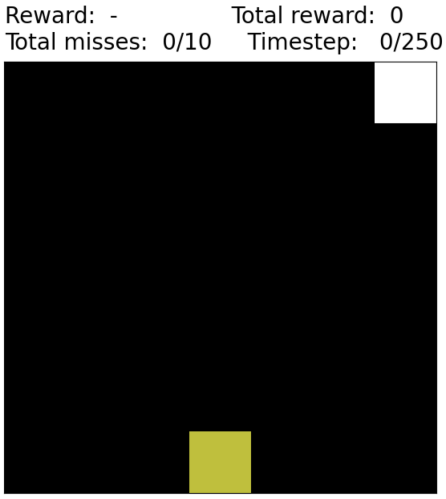

Figure 1: Illustration of Catch environment.

2 Assignment

Part 1: Algorithm variation Implement the following algorithms **from scratch** (it is not allowed to use existing libraries that implement these algorithms such as Stable baselines).

- REINFORCE
- Actor-Critic with:

[1] It is adapted from DeepMind bsuite by Thomas Moerland. You can download the code from BrightSpace.

- Bootstrapping
- Baseline subtraction
- Bootstrapping + baseline subtraction

For each of these algorithms, use **entropy regularization** as an exploration method. Make sure to **tune the strength of the entropy regularization term in the loss**. How does this affect performance? Do not forget to also (at least) tune the learning rate.

Detailed descriptions of all these algorithms can be found in the lecture notes. Use the default Catch environment of size 7x7, with speed at 1.0, and observation type at 'pixel'. Implement your work in Python with either Tensorflow or Pytorch.

Part 2: Environment variation After you finish the above experiments, take the **best** agent and further examine it on different configurations of the environment:

- Vary the size of the environment. How does this affect performance?
- Vary the speed with which the balls drop. How does this affect performance?
- Change the observation type to 'vector'. How does this affect performance? Why can't you use observation type 'vector' with speed > 1.0?
- Combine two changes: use a non-square environment (of width 14, height 7) with a speed of 0.5. Can the agent still catch all balls? What strategy does it learn? Can you think of other interesting variations?

Carefully think about how you can best visualize these questions. **Do not forget to explain/interpret your observations!**

3 Goal

The goal of the assignment is to implement and investigate policy gradient techniques. Questions that you should aim to answer are:

- Do the policy gradients used by REINFORCE indeed suffer from high variance?
- What is the effect of bootstrapping and baseline subtraction, and their combination, within the actor-critic framework on the variance of the policy gradients?
- How do these techniques compare in terms of performance?
- What is the effect of entropy regularization on performance?
- How does the agent perform on tasks with different configurations and why?

As always, make sure to think carefully about the experimental design (what network architecture, what hyperparameters do you use, how to account for randomness, etc.) that you use to answer these questions. Below is a checklist you might want to follow:

☐ Implement REINFORCE, different variants of Actor-Critic.

☐ Design experiments to compare agents you implemented

☐ Try environments with different configurations and report your findings

☐ Answer all listed questions

4 Bonus (optional)

In case you are up for an additional challenge, you can consider:

- Implement a genetic algorithm such as **(CMA-ES)** (Covariance Matrix Adaptation Evolution Strategy) to optimize the policy in a gradient-free way.
- Implement a state-of-the-art policy gradient loss such as the **Clip-PPO** (proximal policy optimization) loss. Search yourself for the correct loss term formulation, which is for example described here here.
- Come up with your own idea! Anything you find interesting is worth exploring.

5 Submission

Make sure to nicely document everything that you do. Your final submission consists of:

- Source code with instructions (e.g., README) that allows us to **easily** (single command per experiment / sub task) rerun your experiments on a university machine booted into Linux (DSLab or computer lab).
- A self-contained scientific pdf report (using the ICML template) of at **at most 8 pages** including figures and references using the provided template. You are not allowed to deviate from this page limit or template. Every page over the limit will result in a loss of 1 point. This report contains an explanation of the techniques, your experimental design, results (performance statistics, other measurements,...), and overall conclusions, in which you briefly summarize the goal of your experiments, what you have done, and what you have observed/learned. Make sure you also list the contributions of your team members.

If you have any questions about this assignment, please visit our lab sessions where we can help you out. In case you cannot make it, you can post questions about the contents of the course on the Brightspace discussion forums, where other students can also read and reply to your questions. Personal questions (not about the content of the course!) can be sent to our email address: rl@liacs.leidenuniv.nl.

The deadline for this assignment is the **7th of May 2023 at 23:59 CET**. For each full 24 hours late, one full point will be deducted (e.g., if your work is graded with a 7, but you are two days too late, you get a 5).

Good luck and have fun! :-)

6 Useful resources

For this assignment you may find the following resources useful:

- Lecture notes by Thomas Moerland
- Definitely go to OpenAIs Spinning Up, and the code repo, which is here.
- Baseline implementations of PPO, and many other algorithms are here. Don't copy code from here, only use it as a reference.
- You could use wandb to log/manage your experiments.