Homework 2

O. Darwiche, E. Oyallon, thanks to G. Charpiat, V. Berger

Exercise

In this project, you are asked to solve a variation of the mountain car problem, using for example a policy gradient algorithm. The mountain car problem is that of a car stuck amidst two mountains. A left or right constant thrust can be applied to the car. The goal is to make the car climb the mountain to its right. The thrust that can be applied is not sufficient for the car to directly go uphill. Momentum has to be gathered to climb high enough.

The following part of the problem have changed:

- The possible actions for your car are continuous: any force value in $[-F_{max}; F_{max}]$ (if you output values outside of this range they will be clamped).
- You receive each step a reward of $-0.1 - \lambda|F_t|^2$, where F_t is the force you last applied
- λ and F_{max} are parameters of the environment that you do not know in advance (and will be randomized on the test bed)
- You receive a reward of 100 for reaching the top of the hill.

This means your agent must find an appropriate balance between getting out of the valley as quickly as possible and not accelerating too much (to save fuel).

You can implement an actor-critic (or any other methods) for this problem, and then, submit your solution on the platform at the following url: `https://competitions.codalab.org/competitions/21854?secret_key=68848036-d6bd-4aa3-a4aa-e85b9ccbd011`

Template description

The template is a zip file that you can download on the course website. It contains several files, two of them are of interest for you: `agent.py` and `main.py`.

`agent.py` is the file in which you will write the code of your agent, using the `RandomAgent` class as a template. Don't forget to read the documentation it contains. Note that you can have the code of your several agents in the same file, and use the final line `Agent = MyAgent` to chose which agent you want to run.

`main.py` is the program that will actually run your agent. You can run it with the command `python main.py`. It also accepts a few command-line arguments:

- `--ngames N` will run your agent for N games against in the same environment and report the total cumulative reward
- `--niter N` maximum number of iterations allowed for each game
- `--batch B` will run B instances of your agent in parallel, each against its own bandit, and report the average total cumulative reward
- `--verbose` will print details at each step of what your agent did. This can be helpful to understand if something is going wrong.
- `--interactive` will train your agent `ngames` times, then run an interactive game displaying informational plots. You need to have `matplotlib` installed to use it.

The running of your agent follows a general procedure that will be shared for all later practicals:

- The environment generates an observation
- This observation is provided to your agent via the `act` method which chooses an action
- The environment processes your action to generate a reward
- this reward is given to your agent in the `reward` method, in which your agent will learn from the reward

This 4-step process is then repeated several times.

Grading

The final performance of your agent will be evaluated by running the following command on a pseudo-random[1] testbed:

```
python main.py --ngames 200 --niter 400 --batch 10
```

Once you think your implementation is good, create a zip file containing your `agent.py` file and the `metadata` file provided in the template, and upload it to the platform. Your score will be computed and you can compare yourself to the rest of the class using the leaderboard. Your grade for this exercise will be based on this score.

[1]This means that uploading two times the exact same code will generate the exact same score