Deep Reinforcement Learning (WMAI024-05) Assignment

Matthia Sabatelli

I. INTRODUCTION

In this Deep Reinforcement Learning (DRL) assignment, you have the chance to implement some of the algorithms that were discussed in class. The assignment is divided into two parts; in the first one (see Sec. II A for further details), you will use a simple benchmark called `Catch`, whereas in the second part (see Sec. II B) you will be free to train another DRL algorithm on - almost - any environment of your choice. In terms of implementation, you can choose whichever DRL algorithm you feel the most comfortable with; however, it is mandatory to train at least two different algorithms; one when it comes to Part II A of the assignment and one when it comes to Part II B of the assignment. The algorithms not only have to be different but also have to be part of two different DRL families of techniques. As we have seen in class, one can categorize different DRL algorithms into three groups: Value-Based Algorithms, Policy Gradient Algorithms, and Model-Based Algorithms (see Fig. 1 for a reminder). Therefore, as a concrete example:

If you decide to solve the `Catch` environment of Part II A with, e.g., the DQN algorithm [5], you will have implemented a Value-Based algorithm. This means that when it comes to Part II B, you must implement either a Policy-Based or a Model-Based algorithm.

In terms of implementation, either when it comes to Part II A of the assignment or when it comes to Part II B of the assignment, you must implement at least one DRL algorithm from scratch. Whether you do this on the `Catch` game (Part II A) or on an environment of your choice (Part II B) does not matter. Note that this means that, on either one of the two DRL benchmarks, you are not allowed to use a `Python` library that already provides you with a working implementation of a DRL agent such as `StableBaseline3; TensorflowAgents, PyTorchDRL, etc ...` You are allowed to use these libraries when tackling the DRL problem which you have not solved with a self-implemented agent. Therefore, as a concrete example:

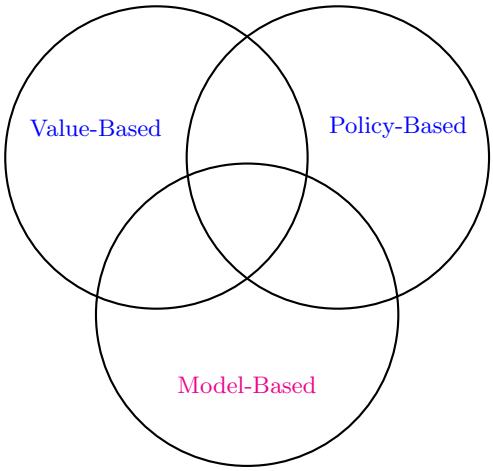

FIG. 1. The three different families of Deep Reinforcement Learning algorithms. You will have to implement at least one algorithm for two out of the three different families. One will be trained on the benchmark described in Part II A of the assignment, while the other one will be trained on the problem you will choose when it comes to Part II B.

If you decide to solve the `Catch` environment of Part II A with, e.g., the DQN [5] implementation provided by the `StableBaseline3` library, that is perfectly fine. This means, however, that when it comes to Part II B, you must implement either a Policy-Based or a Model-Based algorithm from scratch without relying on any DRL library.

The remainder of this assignment is structured as follows: in Sec. II, you will find a description of the DRL environments you can/can't use for training. In Sec. III, you will find a description of what you should hand in as part of this assignment. Finally, in Sec. IV and Sec. V, the evaluation criteria and deadlines are reported.

II. THE DEEP REINFORCEMENT LEARNING TESTBEDS

A. The Catch Environment

The first DRL problem you must solve is the `Catch` environment. The problem is a simple RL task first presented in [4]. It has been widely used within the literature for investigating the performance of DRL algorithms in

FIG. 2. One state of the `Catch` environment.

a fast and computationally less expensive manner than the one required by more popular benchmarks such as the `Atari` games [1, 8]. In the game of `Catch`, an agent controls a paddle at the bottom of the environment, represented by a 21×21 grid, and has to catch a ball falling from top to bottom, which can potentially bounce off walls. At each time step, the agent can choose between three actions: move the paddle one pixel to the right, move it to the left, or do not perform any of the above actions, therefore keeping its paddle in the same position in the grid. An episode ends either when the agent manages to catch the ball, in which case it receives a reward of 1, or when it misses the ball, which naturally results in a reward of 0. Following the design choices presented in [8], the ball has vertical speed of $v_y = -1\, cell/s$ and horizontal speed of $v_x \in \{-2, -1, 0, 1, 2\}$. A visualization of the game is presented in Fig. 2. You are already provided with the implementation of the game of `Catch`, which you can find in Appendix VI. The implementation follows the same design principles as the popular `Open-AI Gym` library. If you are unfamiliar with the library, please check out its official documentation page `https://gymnasium.farama.org/`. Next to that, note that some processing operations are already applied for you such that what is returned by the environment resembles, as much as possible, the way the input space of the `Atari` games were pre-processed in [5]. Please do not modify the code provided in Appendix VI for any reason; the way the environment is implemented guarantees successful training [6]. If you are missing the `scikit-image` library, you can install it with the `conda install scikit-image` or `pip install -U scikit-image` commands.

B. Choose Your Own Environment

In the second part of the assignment, you are free to choose - almost - whatever DRL problem you like best: the aforementioned `Open-AI Gym` library [2] provides some exciting and popular test beds to choose from, and so does the `MuJoCo` simulator [7]. Note, however, that some of the proposed environments can be computationally expensive. You can also design your own DRL environment as long as it will be appropriately described in the final paper (see Sec. III for further details). However, there are a few environments that you are not allowed to use due to their simplicity or the many available online resources which would provide you with an already working implementation of the DRL algorithm.

The following environments/benchmarks are **forbidden**:

- `Cartpole` both the `-v0` version as well as the `-v1` version.
- `Acrobot`
- `Mountain-Car` both the discrete version as well as the continuous version.
- `Lunar-Lander`
- `Viz-Doom`

III. PROJECT DELIVERABLE

To complete the assignment, you will have to write an eight pages max report in the form of a scientific paper where you will walk the reader through the different DRL algorithms that you have implemented for the problems described in Sec. II A and Sec. II B. The paper should - at least - i) give the reader a general introduction to the overall content of the report, ii) provide detailed explanations about the DRL environments used as test beds, iii) thoroughly explain which algorithms have been implemented, and iv) critically present and discuss the obtained results. You will have to submit the paper on Brightspace together with the code used for your experiments. The paper should follow the popular ICML LaTeXtemplate, which you can find here: `https://icml.cc/Conferences/2020/StyleAuthorInstructions`. The eight-page limit does not include Appendices nor References.

IV. EVALUATION CRITERIA

This written assignment accounts for 40% of your final grade. There will also be a presentation slot allocated for each group where you will have the chance to present your results in front of your peers. This presentation will account for 10% of the final grade, and more detailed information about it will follow. Your written report will be evaluated based on the following three different criteria:

1. Presentation: this criterion accounts for 25% of the grade and will consider how well the overall report looks like: "does it contain spelling mistakes?"; "is the writing redundant or not-concise?"; "do you

respect the given LaTeXtemplate and the maximum page limit?", "is the report well structured?"; etc ...

2. Clarity: this criterion also accounts for 25% of the grade. It considers how understandable and precise your final report is: "are the different DRL algorithms explained correctly?"; "are the different DRL environments described in detail?"; "is every variable used in the equations well defined?"; "is every algorithm described formally?"; "is the overall experimental setup well described, and does it contain enough details that allow for reproducibility?"

3. Results: this criterion accounts for 50% of the grade. The report must show proof that you successfully trained at least two DRL algorithms. This means that you will have to report learning curves, possibly averaged over different training runs, that allow you to statistically assess the agents' performance. These results should be reported in clear plots with understandable and readable captions, axes, and titles. While the minimum requirement is that of successfully training two agents that are part of the family of DRL algorithms presented in Fig. 1, you can significantly improve the grade of this criterion by training more than two agents. If you need inspiration on how to report and interpret robustly the performance of your agent I refer you to Chapter 9 of [3].

As mentioned in Sec. III, next to the written report, you will also have to submit your code; this one will, however, not be graded. Yet, it might be checked when the final report does not contain enough details to understand the experiments performed or for plagiarism. Note that if you use code from a third party, **you must** acknowledge it in the report.

V. DEADLINES

The deadline for handing in the final paper is Thursday, the **15th of June at 12:00 a.m.** However, to ensure that you spend an $\approx$ equal amount of time on both parts of the assignment (Sec II A and Sec. II B), you will have to submit the results obtained on the `Catch` environment by the **16th of May**: more instructions about the exact submission details will follow.

[1] Samy Aittahar, Raphaël Fonteneau, and Damien Ernst. Empirical analysis of policy gradient algorithms where starting states are sampled accordingly to most frequently visited states. *IFAC-PapersOnLine*, 53(2):8097–8104, 2020.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[4] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[6] Matthia Sabatelli and Pierre Geurts. On the transferability of deep-q networks. In *Deep Reinforcement Learning Workshop of the 35th Conference on Neural Information Processing Systems*, 2021.

[7] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[8] Jos van de Wolfshaar and Lambert Schomaker. Deep learning policy quantization. 2018.

VI. APPENDIX

```
from skimage.transform import resize
import random
import numpy as np

class CatchEnv():
    def __init__(self):
        self.size = 21
        self.image = np.zeros((self.size, self.size))
        self.state = []
        self.fps = 4
        self.output_shape = (84, 84)

    def reset_random(self):
        self.image.fill(0)
        self.pos = np.random.randint(2, self.size-2)
        self.vx = np.random.randint(5) - 2
        self.vy = 1
        self.ballx, self.bally = np.random.randint(self.size), 4
        self.image[self.bally, self.ballx] = 1
        self.image[-5, self.pos - 2:self.pos + 3] = np.ones(5)

        return self.step(2)[0]


    def step(self, action):
        def left():
            if self.pos > 3:
                self.pos -= 2
        def right():
            if self.pos < 17:
                self.pos += 2
        def noop():
            pass
        {0: left, 1: right, 2: noop}[action]()


        self.image[self.bally, self.ballx] = 0
        self.ballx += self.vx
        self.bally += self.vy
        if self.ballx > self.size - 1:
            self.ballx -= 2 * (self.ballx - (self.size-1))
            self.vx *= -1
        elif self.ballx < 0:
            self.ballx += 2 * (0 - self.ballx)
            self.vx *= -1
        self.image[self.bally, self.ballx] = 1

        self.image[-5].fill(0)
        self.image[-5, self.pos-2:self.pos+3] = np.ones(5)

        terminal = self.bally == self.size - 1 - 4
        reward = int(self.pos - 2 <= self.ballx <= self.pos + 2) if terminal else 0

        [self.state.append(resize(self.image, (84, 84))) for _ in range(self.fps - len(self.state)
                                                                        + 1)]
        self.state = self.state[-self.fps:]

        return np.transpose(self.state, [1, 2, 0]), reward, terminal

    def get_num_actions(self):
        return 3

    def reset(self):
        return self.reset_random()

    def state_shape(self):
```

```
        return (self.fps,) + self.output_shape


def run_environment():
    env = CatchEnv()
    number_of_episodes = 1

    for ep in range(number_of_episodes):
        env.reset()

        state, reward, terminal = env.step(random.randint(0,2))

        while not terminal:
            state, reward, terminal = env.step(random.randint(0,2))
            print("Reward obtained by the agent: {}".format(reward))
            state = np.squeeze(state)

        print("End of the episode")


if __name__ == "__main__":
    run_environment()
```