Reinforcement Learning Lab

Final Activity

Davide Corsi and Alberto Castellini

University of Verona
email: davide.corsi@univr.it

Academic Year 2022-23

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Assignment

The objective of the final activity is to train an agent to solve a popular DRL problem Mapless Navigation. You will find the code here:

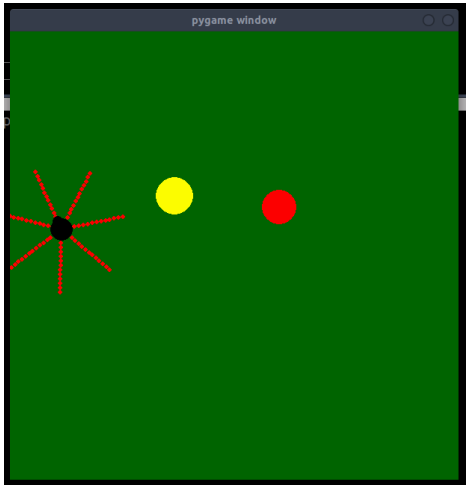
RL–Lab/lessons/final_activity.py

Inside the file, you will find a template with the code that you can complete with one of the algorithms implemented during the course. You will find also a class called OverrideReward that you should modify to find the best reward function to solve the problem, in particular, the method *step*. The class and method to complete are:

- **class OverrideReward()**
- **def step(self, action)**

Notice that other changes should be made in order to adapt the algorithms to a different environment (e.g., network shape and hyperparameters); and some additional function implemented (e.g., **createDNN**, **training_loop**, and **A2C**).

Environment: Mapless Navigation



- The Mapless Navigation problem is a classic benchmarking task for reinforcement learning. The objective for the agent (the black circle) is to reach the goal position (yellow circle) while avoiding collision with the obstacles (red circle).
- The red dotted lines represent the obstacles sensors of the agent.
- The **state** of the environment is represented as a tuple of $2 + 7$ values: *Heading*, *Distance* to the goal, and *Lidars Sensors*.
- The **actions** allowed in the environment are 3: *action 0 (accelerate)*, *action 1 (turn left)* and *action 2 (turn right)*.

State Space

1. The first element of the state is the **heading**, which represents the angle with respect to the goal position. This value ranges from $[-1, 1]$; where 0 means that the goal is right in front of the agent, ± 1 that the goal is behind it, 0.5 that the goal is on the left, and 0.5 on the right.
2. The second element of the state is the **distance** from the goal position. The value is normalized in the range $[0, 1]$, where 0 means that the agent already reached the goal and 1 that the goal is at the maximum distance.
3. The other elements of the state (from 2 to 9) represent the **collision sensors**. The seven sensors start from the front of the robot and turn in a clockwise sense (e.g., sensor 0 checks for obstacles in front of the robot, sensor 1 checks for obstacles at ≈ 51 degrees on the right, etc.). The value returned by each sensor ranges from 0 to 1, where 0 means that an obstacle is close to the agent and 1 that there are no obstacles in the corresponding direction.

Override the Original Reward

To modify the reward function, in the suggested code we exploit the Gymnasium Wrappers to override the step function. In the provided template the following code is already implemented:

```
# Extract the information from the observations
old_heading, old_distance, old_lidars = \
    self.previous_observation[0], self.previous_observation[1], \
    self.previous_observation[2:]
heading, distance, lidars = observation[0], observation[1], observation[2:]
# Exploiting useful flags
goal_reached = bool(info["goal_reached"])
collision = bool(info["collision"])
```

In addition to the observations, the environment provides two useful boolean flags (that are not considered part of the state): **info["goal_reached"]** indicates when the agent reaches the goal; **info["collision"]** indicates when the agent touches the obstacle. Both of these states are terminal.

Additional Information

1. The original reward function is *sparse*, specifically a reward bonus of 1 is assigned when the agent reaches the goal, 0 otherwise.
2. An episode terminates after 250 steps if the agent does not reach the goal and does not collide with an obstacle. Other termination criteria are (i) *collision with an obstacle* or (ii) *goal reached*.

Reward Design

Not all the information in the state space are necessary to write the reward function. Try to ask yourself *what is the real objective of the task?* And *what kind of information would you need to solve the problem?*

Consult the previous slides!

Remember that you can find useful information from the slides of the previous lessons! In particular from lesson_10 and lesson_11.

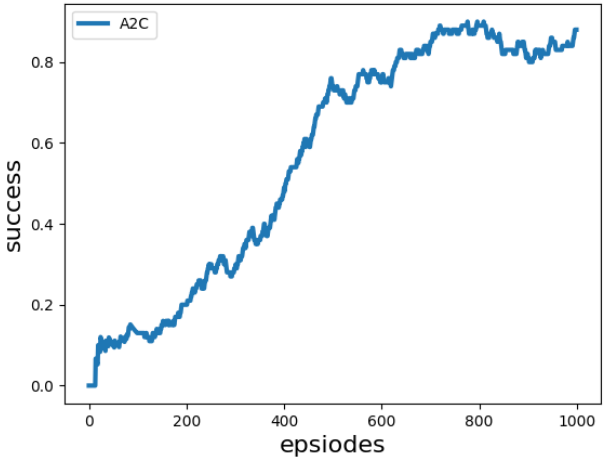
Expected Results

Monitoring Metric

For this problem, the metric to consider is the **success rate**; i.e., the number of successful trajectories in the last 100 episodes. A template is already provided to compute and return this value.

Results

Note that obtaining this result requires time. You can consider the task solved if the agent reaches a success rate greater than 70/100,



What you should submit

To complete the test, you should upload the following material in the Moodle folder *"Partial exam 2"* the following material in a zip file:

1. the developed code (i.e., id_surname.py)
2. the resulting plot with the result (i.e., id_surname.png)
3. the trained policy (i.e., id_surname.h5)