Reinforcement Learning Lab

Lesson 1: Policy Iteration and Value Iteration

Davide Corsi and Alberto Castellini

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

Academic Year 2022-23

Initial Setup Process - 1

The first step for the setup of the laboratory environment is to install miniconda, a light version of the Conda package manager.

- Download the *Miniconda* package manager: `https://docs.conda.io/en/latest/miniconda.html`
- Install *Miniconda*
  - On *Linux* run *Miniconda3-latest-Linux-xx.sh*, remember to provide the executions permissions to the file.
  - On *Windows* double-click the installer, in the installation phase, ensure to install *Anaconda Prompt* and use it for the next steps.

Virtual Environment

For python virtual environments' users (*venv*), the Miniconda installation can be avoided.

Initial Setup Process - 2

The second step is to download the official laboratory repository and create the working environment. The following commands must be run in the *Linux Shell*, or in the *Anaconda Prompt* on Windows.

- Clone the official Lab repository:
  git clone https://github.com/d-corsi/RL-Lab
- Create the environment and install the required packages:
  conda env create -f RL-Lab/tools/rl-lab-environment.yml
- Before running the scripts, remember to activate the environment:
  conda activate rl-lab

Git Installation

The installation of Git may be required, for Linux users just run *sudo apt-get install git*. For Windows users, refer to the official page `https://git-scm.com/downloads`.

First Tutorial

The README file on the repository (link) contains a simple tutorial on the basic libraries and tools we will use in the next lessons, in particular:

Numpy

NumPy is the fundamental package for scientific computing with Python, think of it as a MATLAB-like environment for Python.

OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms, a standardized set of tools and instructions to work with environments.

Keras

Keras is a high-level neural networks APIs that implements simple functions to create, train and modify neural networks.

Assignments Structure

In general, the requirement for a laboratory lesson is to complete the partial python code provided, for example:

```
RL-Lab/lessons/lesson_n_code.py
```

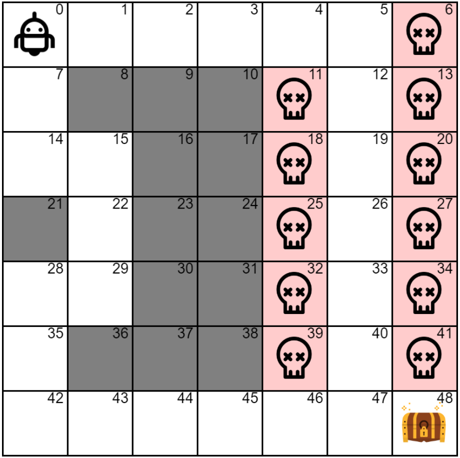
The **main** function is always provided in a Python script. Once the code is completed, students have to run the script to obtain all the results in the output on the console. The expected results can be found in the *.txt* file corresponding to the lesson, for example:

```
RL-Lab/results/lesson_n_results.txt
```

Disclaimer

Given the stochasticity of the algorithms that we will run in the course, the results can be slightly different from the ones reported in the *.txt* file, the relevant point is the general trend.

Environment Description



Grid-World environment, the goal for the robot is to reach the target (treasure) while avoiding the terminal states (death). Dark cells represent walls that can not be crossed.

All cells return a reward:

- **+1** for the target position (state 48)
- **-1** for the terminal cells (death)
- **-0.1** for each empty step

Today Assignment

In today's lesson, we implement the value iteration and policy iteration algorithms in Python. In particular, the file to complete is:

```
RL-Lab/lessons/lesson_1_code.py
```

Inside the file, two functions are only partially implemented, the objective of this lesson is to complete them.

- **def value_iteration()**
- **def policy_iteration()**

Expected results can be found in:

```
RL-Lab/results/lesson_1_results.txt
```

Pseudocode - Value Iteration

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figure: Pseudocode for the value iteration algorithm, the implementation is from the Russell and Norvig book: *Artificial Intelligence: A Modern Approach*

Pseudocode - Policy Iteration (a)

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Figure: Pseudocode for the policy iteration algorithm, the implementation is from the Russell and Norvig book: *Artificial Intelligence: A Modern Approach*

Pseudocode - Policy Iteration (b)

The *policy evaluation* of the policy evaluation algorithm implements the following function:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s)) U_i(s') \, .$$

Figure: Policy Evaluation function.

Suggestion:

In both the assignments, the update functions require to discount the future reward (e.g., $r + \gamma \cdot future$). Remember that for the terminal states there is no future! Update only with r in such cases.