Departamento de Informática
Universidad Carlos III de Madrid

Bachelor in Data Science and Engineering

Machine Learning I

Course 2020-2021

Assignment 2

Reinforcement Learning

April 18, 2022

1 Introduction

In this assignment, you will apply the *Q-learning* algorithm in the domain of the Pac-Man, to build an agent that works automatically in the greatest possible variety of labyrinths. The objective of the agent will be to maximize the score obtained. To do this, we will use reinforcement learning techniques.

2 Tasks to be performed

The tasks to be carried out in this assignment are divided into different phases (selection of state information and reward function, construction of the agent, and evaluation) which are detailed below:

Phase 1. Selection of the state information and reward function

For the agent to learn, you will use learning tuples in the form of (*state_tick*, *action*, *state_next_tick*, *reward*), where *state_tick* contains information about the current state of the game, *action* is the action executed by Pacman in that state, *state_next_tick* is the state reached after applying *action* in *state_tick*, and *reward* is the reward obtained when performing the transition. To do so, you should take into account the following aspects:

1. Select the attributes to be used to represent a game state. This information should be generic and not linked to particular mazes, as it is intended that Pac-Man plays well in as many mazes as possible. Also, the fewer attributes you select the better, and if they are nominal with few values the better, since the size of the Q-table will depend on this selection.
2. The tuples *state_tick* and *state_next_tick* has the same attributes, but representing different ticks.
3. Design a reward function that allows Pacman to maximize the score of a game.

Phase 2. Generation of the agent

1. Implement a new class in *busterAgents.py* called *QLearningAgent*. **In AulaGlobal, you can download of the code the Q Learning algorithm implemented in the Pacman code. You have to apply the changes in bustersAgent.py - QLearningAgent class - in methods: update(), computePosition() and getReward(). The objective is to implement the state corresponding to the proposed solutions, access to Q table and reward function.**

2. Implement the agent functionality of the tutorial 4 in this new class so that a Q-table is built and updated according to the Q-Learning algorithm. This table will have as many rows as number of states you have decided to use, and as many columns as actions Pac-man can execute. For example, if we select two attributes to represent the state: the relative position of the closest ghost with respect to Pac-man, and its distance, we have a nominal attribute with 8 possible values (*up, left, right,..., up, left*) and we have a numerical attribute whose range will depend on the size of the biggest maze we play in. Let's suppose that this range goes from 1 to 10, where 10 is the maximum distance a ghost can be. Then, the number of rows in our Q table will be $8 \times 10 = 80$ rows, which are all the possible combinations of values of those two attributes. Therefore, you can see that the larger the number of attributes and the range of these, the larger the size of the Q-table. Therefore, it is recommended to have few attributes and to have a manageable size of the Q-table that favors code debugging and learning speed.

3. Select the best value for the α, ϵ and γ attributes.

4. The operation of the agent should be as follows:

   - At the beginning of the execution, the agent reads an existing Q-table through a method called *readQTable*, setting its result in the proper variables (as in Tutorial 4).
   - On each tick, you have to build a tuple in the form (*state_tick*, *action*, *state_next_tick*, *reward*) with the information you previously considered. Note that you should perform a deferred behavior in the update process similar to the one you used in Assignment 1.
   - These tuples are sent as parameters to an *update* method that updates the Q-table, as you did in Tutorial 4.
   - On each tick, when Pacman wants to execute an action, you will use the ϵ-*greedy* strategy you employed in Tutorial 4. This means that Pacman will execute the action with $maxQ(s, a)$ value with probability ϵ and a random action with probability $1 - \epsilon$.
   - When a game ends, write the Q-table using the *writeQTable* method. By doing this, when a new game starts, Pacman will use this pre-existing Q-table.

Suggestions:

1. For the construction of the agent it is recommended to follow a process of incremental development, in which first simple labyrinths are addressed, and then gradually increase the difficulty. To do this, in this practice there are 5 labyrinths available: *labAA1.lay, labAA2.lay, labAA3.lay, labAA4.lay, labAA5.lay.* It is recommended to focus first on *labAA1.lay*, and build an agent that works perfectly in this labyrinth. Once obtained, build an agent that works well in *labAA1.lay, labAA2.lay.* And then build an agent that works well in the first three mazes. And so on, until you get an agent that works well in all the mazes.

2. During the learning process, it may be necessary to evaluate what the agent has learned so far by eliminating the random exploration of actions. To do this, between games, you can set the values of ϵ and α to 0 and launch a game with these parameters. If the agent still does not work well, you can reset the values ϵ and α to their previous values to continue with the learning.

3. Also during the training, you can check if the updates in the Q-table are being done correctly. To do this, you can open the file that stores the table and check that a tuple of experience actually leads to updating the corresponding cell.

Phase 3. Evaluation of the agent

1. Evaluate your agent against as many agents as possible using different maps and ghost's configurations.

2. Report both quantitative and qualitative results on the agents' behavior.

3 Files to Submit

All the lab assignments **must** be done in groups of 2 people. You must submit a .zip file containing the required material through Aula Global before the following deadline: **May 13th at 8:00**. The name of the zip file must contain the last 6 digits of both student's NIA, `i.e., assignment2-123456-234567.zip`

The zip file must contain the following files:

1. A **PDF** document of **no more than 20 pages** with:
   - Cover page with the names and NIAs of both students.
   - Explanation of each decision you made when defining the states and the reward function.
   - The experimentation carried out at each phase.
   - Conclusions.
2. The **code** of your automatic agent. You should put *all* the code files within a .zip file. If the file is extracted and the command `python busters.py -p QLearningAgent` is executed, it must run the Pacman agent you want to submit.

Please, **be very careful and respect the submission rules**. The weight of this assignment in the final mark of the course is **2 points**. The following rubric details the marking criteria:

- Definition of states and reward function (0.5). We will consider the analysis of different alternatives, as well as their implication on the size of the Q-table and the behavior of the agents.
- Evaluation of the agents (0.7). We will consider the proper description of the experimental setting, the variation of parameters, and the comparison between different agents, among others criteria.
- Performance of the resulting agent (0.3). We will automatically evaluate your agent in a set of maps, observing the score it gets.