Lab session 1: Intelligent Agents

Pierre-Alexandre Murena

April 4, 2024

1 Preliminary information

1.1 Prerequisites

This lab session, as well as the following one in this course, will be done in Python. It will require the use of **python 3.7 or newer** and of the packages **numpy** and **matplotlib**.

1.2 How to get the source code for the lab sessions

All the code used during the lab sessions can be found in the following GitHub repository: `https://github.com/ppaamm/TUHH-AICourse`. The material for the sessions is made available after the lecture.

2 The monkey and the banana environment

In this session, we explore the problem of the monkey and the banana introduced in the lecture. In this problem, the agent is a monkey whose goal is to grab a banana hanging from the ceiling. The monkey can reach the banana by climbing upon a box.

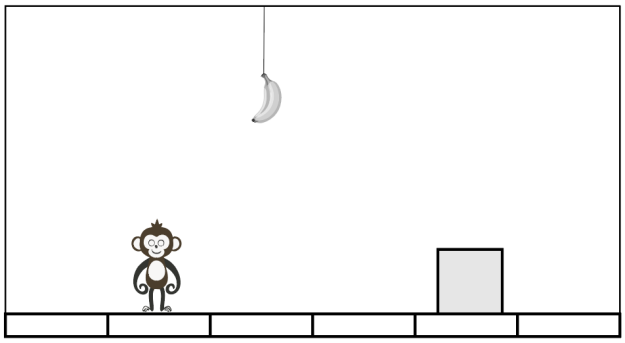

Figure 1: The problem of the monkey and the banana

2.1 First contact with the environment

All the tools related to the environment are implemented in the file `banana_environment.py`. For a first contact with the environment, the three classes to consider are:

- `MonkeyBananaState`: Implementation of a state of the environment, namely the horizontal position of the box (integer), the horizontal position of the banana (integer) and the vertical position of the monkey (Boolean). Note that the position of the monkey describes whether the monkey is on the box (value equals `True`) or on the floor (value equals `False`).

- `MonkeyBananaAction`: Enumeration of the potential actions of the monkey. Note that this set of actions is fixed, but that in practice all of them won't be necessarily available to the agent. Determining which actions are available in which state is the responsibility of the environment class itself.

- `MonkeyBananaEnvironmentTask`: Main implementation of the environment task. This class contains all the methods relative to the task and to the interaction between the agent and the environment.

The first two important methods to consider are the constructor and the visualization (`visualize`). The constructor instantiates an environment of a given size (i.e. a given number of positions on the horizontal axis), given by the `room_size` argument, and the initial position of the banana and of the box. The monkey is supposed to be initially on the floor. The visualization method provides an ASCII representation of the room, where the banana is represented by the character '`b`', the monkey by the character '`m`', the box by the character '`x`' and the floor positions by some '`x`'s.

Question 1. Get familiar with the environment, by instantiating an environment of size 10 in which the box is located at $x = 2$ and the banana at $x = 7$. Visualize this environment.

In any state the list of available actions can be retrieved using the `available_actions` method. Once the agent selects its action, it can perform it within the environment using the `perform_action` method. The method outputs a Boolean which indicates whether the monkey successfully grabbed the banana.

Question 2. Using the environment instantiated in Question 1, write a series of actions that should lead the agent to grabbing the banana. Check that it is successful by implementing it. Using the `performance` method, compute the score of the agent.

Some series of actions are sub-optimal, in the sense that they lead to the goal but with a potentially high number of unnecessary actions. For instance, the monkey can start moving the box to the left then to the right. This is sub-optimal because it leads back to the initial state after two actions which are then useless.

Question 3. Is the sequence of actions found in the previous question optimal or sub-optimal? Is there a unique optimal sequence of actions? Prove the result.

2.2 Fully vs partially observable environment

A third essential component of a task is the perception, i.e. what of the total environment, can be observed by the agent. This is implemented in method `perceive`, which is left abstract in the `MonkeyBananaEnvironmentTask` class. Two variants are proposed: a variant for fully observable environments, and a specific partially observable environment.

In a fully observable environment task, the agent perceives the entire state of the environment. This is implemented within the `MonkeyBananaFOEnvironmentTask`, which is a child class of `MonkeyBananaEnvironmentTask`. Here the `perceive` method returns the full state.

In a partially observable environment task, the agent does not perceive the exact state. We assume that, in this environment, the monkey can smell the banana when on top of the box. A full description of the probability is given in the last part of the lab session. The observation then describes the position of the box, the position of the monkey, and whether the monkey can smell the banana. It is implemented in class `MonkeyBananaPartialObservation`. The class `MonkeyBananaPOEnvironmentTask` is an implementation of the corresponding environment task.

3 Random agent

The simplest possible agent is a fully random agent. Such an agent makes its decision uniformly at random at each time step, among the available actions.

3.1 Theoretical analysis

We first propose to show that the agent will eventually grab the banana with probability 1.

Question 4. What is the probability that the agent performs the optimal sequence of actions? Is this agent rational?

We will now propose a lower bound of the probability that the agent reaches the banana before time t. To do so, we will decompose the trajectory into several segments.

Question 5. Knowing that the box starts at position $x \in \mathbb{N}$ and that the banana is at position $b \in \mathbb{N}$, what is the optimal trajectory? What is the probability that the random agent plays this exact trajectory?

Question 6. We denote by T the length of this trajectory. Show that for all $t < T$, the probability of victory is 0. What is the probability of victory at time $t = T$? Propose a lower bound and an upper bound of this probability that does not depend on x and b (but can depend on the length L of the environment).

In other words, if we wait until time T, either the agent just grabs the banana, or we know that the agent did not grab the banana. We know the exact probability of these two events. In case the agent did not grab the banana, we can run the process again.

We now use the following notations. We call a **state** the tuple $s = (x, m)$ where $x \in \mathbb{N}$ is the position of the box and $m \in \{0, 1\}$ is the position of the monkey (0 = down, 1 = up).

We designate by s_t the state at time t. We designate by $v_t \in \{0, 1\}$ the victory status. We note by T_1 the length of the optimal trajectory starting from s_0. In Question 6, we have proven the existence of $0 < m, M < 1$, depending on the length L of the environment only, and such that

$$m \leq p(v_{T_1} = 1 \mid s_0) \leq M. \tag{1}$$

Starting from state s_{T_1}, we can use the same reasoning: there is a unique optimal trajectory leading to victory. We call T_2 the time step where this trajectory ends.

Question 7. At which time steps $t \leq T_2$ do we have $p(v_t = 1 \mid s_0) > 0$? We denote by p_2 the probability that the agent grabs the banana at latest at $t = T_2$. Show that

$$p_2 = p(v_{T_1} = 1 \mid s_0) + p(v_{T_1} = 0 \mid s_0)\ p(v_{T_2} = 1 \mid v_{T_1} = 0, s_0) \tag{2}$$

Question 8. Compute lower and upper bounds for $p(v_{T_2} = 1 \mid s_1)$. By marginalizing over s_1, show that

$$m \leq p(v_{T_2} = 1 \mid v_{T_1} = 0, s_0) \leq M. \tag{3}$$

We can continue the process. For all k, we denote by T_k the process where the k-th optimal trajectory ends (meaning that the first $k - 1$ trajectories were not optimal; in case the optimal trajectory was reached already, we simply take $T_k = T_{k-1} + 1$). We denote by p_k the probability that the banana was grabbed before time step T_k.

Question 9. Show that

$$p_{k+1} = p_k + (1 - p_k)\ p(v_{T_{k+1}} = 1 \mid v_{T_1} = \ldots = v_{T_k} = 0, s_0). \tag{4}$$

and deduce that (p_k) converges toward a finite limit $\ell \in (0, 1]$. Observing that the convergence of (p_k) implies that $|p_{k+1} - p_k| \to 0$ and using the same reasoning as in Question 8, show that $\ell = 1$.

3.2 Experimental section

Now that we have shown theoretically that this solution is actually finding a solution (in infinite time), we will observe the behaviour of the agent in practice.

Question 10. Run 100 random agents and measure the number of steps needed to grab the banana. Compute the average number of steps and plot the distribution. How many agents do find the optimal trajectory?

4 Simple reflex rule-based agent

A **rule-based agent** takes actions by following a predefined set of (deterministic) rules. Such an agent is implemented in the class `RuleBasedAgent`. When choosing the actions, the rule-based agent first observes the environment, and then obeys a series of rules.

Question 11. Execute the currently implemented rules. What is the behaviour of the agent. Implement a set of rules that guarantee the agent to follow the optimal trajectory.

The main problem with rule-based systems is that they require to have a predefined set of rules. In many situations, such rules are not easy to define for the user. This is the reason why most AI techniques aim to be autonomous enough and to avoid relying on some predefined knowledge of the user.

5 Planning agent

The planning agent implemented in class `PlanningAgent` explores several potential paths to get to the banana. At each time step, the agent simulates the result of all potential actions it can make, which results in a foreseen future environment (variable `env_copy`). The planning is then called recursively from this new state. Such a search is called **depth-first search**.

Question 12. Launch an instance of the planning agent. You can observe that the search enters an infinite loop. Can you explain why? Fix the code to make the agent avoid infinite loops.

6 A Bayesian rule-based agent

We consider a variant of the environment where the agent has partial observability: It knows the position of the box and its own position, but ignores the position of the banana. Otherwise, the rules of the environment are exactly the same: in particular, the monkey can grab the banana when on top of the box and under the banana.

Question 13. Which of the previous three agents can work in this variant?

We propose a new agent adapted to this setting of partial observability. In this context, the agent does not know the location of the banana but has to guess it. In practice, this means that the agent will observe the environment and build some **belief** about the location of the banana. A belief is defined as a probability distribution over the states of the environment. Here, since the location of the box and of the monkey are known, there is then a finite number of states $s_1, \dots, s_L$ which correspond to each possible position of the banana (in state s_i, the banana is located at i). Consequently, the belief sums up to a probability distribution over the location of the banana. It is then a vector π of dimension L such that

$$\sum_{i=1}^{L} \pi_i = 1 \tag{5}$$

If the agent knows that the banana is located in b, it would correspond to having a belief π such that $\pi_b = 1$ and $\pi_i = 0$ for all $i \neq b$. Conversely, if the agent has absolutely no idea where the banana is located, its belief would be uniform: $\pi_1 = \ldots = \pi_L = \frac{1}{L}$.

Question 14. Given a state s_i, it is easy to model the observations o made by the agent, using probability $p(o \mid s_i)$. This however does not provide any (direct) information on the probability of s_i. Using Bayes rule and assuming the belief π_i, express the quantity $p(s_i \mid o)$ for the agent, i.e. the belief that the environment is in state s_i, as a function of $p(o \mid s_1), \dots, p(o \mid s_L)$ and π only.

At each step t, after doing an action, the agent gets a new observation o_t and updates its belief by taking $\pi \leftarrow p(s_i \mid o_t)$. This belief update method is called **Bayesian belief update**. In the expression found in Question 14:

- The term π_i is called the **prior**: it measures the belief that the agent has in each state *before* any new observation;
- The term $p(o \mid s_i)$ is called the **likelihood**: it measures how *likely* every possible observation is according to each state;
- The term $p(s_i \mid o)$ is called the **posterior**: it measures the belief that the agent has in each state *after* it got the new observation

We consider that the monkey cannot see the banana, but only smell it when on top of the box. When the monkey is on the box, it can smell the banana with a probability $\frac{1}{1+\alpha|x-b|^2}$ where α is some sensitivity parameter, x is the position of the box and b is the position of the banana.

Question 15. Implement the Bayesian belief update for the `BayesianAgent` class, in the method `update_belief`, with the proposed likelihood.

Once the agent updates its beliefs, it must choose an action. Several options are possible here, but we choose to implement a simple strategy:

- If some state s_i is associated to a belief $\pi_i > 0.7$, move the box to this position, climb and try to get the banana.
- Otherwise move the box to any position with belief $\pi_i > 0.2$ and climb.

Question 16. Implement this agent. Don't forget to change the belief update to take into account that $\pi_i = 0$ in case the monkey tried to grab the banana but did not succeed.