Exercise session 2: Wumpus

Guillaume Charpiat, Victor Berger

January 18, 2018

Several questions are in this subject. They are meant to guide you through the exercise, but do not require a written answer. Only your agent will have to be submitted to the scoring platform.

Exercise

The Wumpus world is a well-known toy problem in artificial intelligence popularized by the reference book of Russell and Norvig, 2003. The game consists in a grid world where an agent is looking for a treasure while avoiding the deadly Wumpus and some holes. We assume that the time is discretized and that the agent can take only one action per time step. The agent can explore the grid by moving in the four cardinal directions. Also, it has a flashlight with a limited number of power units that can be used to kill the Wumpus. For the project, the Wumpus world will be our benchmark problem for studying reinforcement learning (RL) algorithms. Moreover, these algorithms are general enough to be applied to different problems (environment).

In the RL framework, we define an environment, which specifies the information to be used by the agent to take some actions. For the Wumpus world, the environment is partially observable since the locations of the holes, the Wumpus and the treasure are unknown to the agent. However, the environment provides some signals to the agent:

- if the agent is adjacent to the Wumpus, it receives a *smell* signal
- if the agent is adjacent to the hole, it receives a *breeze* signal
- the location of the agent is deterministically determined by the initial position and the actions
- the number of power units is known

At each time step, the agent can choose one of 8 different actions: Up, $Down$, $Left$, $Right$, $FlashUp$, $FlashDown$, $FlashLeft$ and $FlashRight$. Attempting to move through the bounding walls of the arena results in nothing happening.

Each time step can lead to 5 different outcomes, in terms of reward for your agent:

- **The treasure is found:** reward +100 and the game ends
- **You kill the Wumpus:** reward +10
- **The Wumpus catches you:** reward −10 and the game ends
- **You fall into a hole:** reward −10 and the game ends
- **Nothing happens:** reward −1

Question 1 Why is it interesting to give a negative reward for the "nothing happens" event ?

The observations available to the agent can be summed up as the following observation space:

$$\mathcal{O} = (X, Y, S, B, F)$$

where X and Y are the coordinates of the location of the agent, S is a boolean value indicating the presence of the *smell* signal, B is a boolean value indicating the presence of the *Breeze* signal, and F the number of remaining charges for the flashlight.

This observation and some previously saved information by your agent can form its state s.

Question 2 Run the provided random agent. You can activate the `--verbose` flag to see a map of the world printed to observe the dynamics. What do you observe on its cumulative reward?

During this exercise, we will study RL algorithms for learning the optimal policy that maps each state to an action. For a given state s, we are interested in the expected reward $\mathbb{E}_{p(r|s,a)}[r]$ where $p(r|s,a)$ is the probability to obtain a reward r given that the agent takes action a in state s. However, $p(r|s,a)$ is unknown to the agent, but we can still approximate it with empirical averages. For this, each time the agent is in state s, it chooses an action a, records the observed reward (history) and updates the average $Q(s,a)$.

In the contextual bandit framework, each state s is considered as a different context, leading (in the simplest framework) to as many independent bandit problems as there are states. The arms of the bandit s are the possible actions the agent can perform in that state s.

Question 3 Implement a contextual bandit algorithm with one of the policies we studied (optimistic, ϵ-greedy, softmax or UCB), and display its cumulative rewards. What is the limitation of modeling the problem with contextual bandits? Possibly consider different state spaces such as $\mathcal{S} = (B, S, F)$, $\mathcal{S} = (A_{previous}, B, S, F)$ (i.e. remembering the previous action) and $\mathcal{S} = (X, Y, B, S, F)$ and study the relation between the grid size, the number of states, the amount of history, ... and the performance of the policy. In order to obtain statistics on the performances, you can run the training algorithm many times using the `--batch` argument.

Question 4 Implement the Q-learning algorithm. Compare with the contextual bandits. When you are satisfied with it, upload your agent on the platform using the link provided on the website.

Template description

The template is a zip file that you can download on the course website. It contains several files, two of them are of interest for you: `agent.py` and `main.py`.

`agent.py` is the file in which you will write the code of your agent, using the `RandomAgent` class as a template. Don't forget to read the documentation it contains. Note that you can have the code of your several agents in the same file, and use the final line `Agent = MyAgent` to chose which agent you want to run.

`main.py` is the program that will actually run your agent. You can run it with the command `python main.py`. It also accepts a few command-line arguments:

- `--ngames N` will run your agent for N games against in the same environment (the same initial grid setup) and report the total cumulative reward
- `--niter N` maximum number of iterations allowed for each game
- `--batch B` will run B instances of the game (B different Wumpus worlds, i.e. with the same rules but different initial grid setups)
- `--verbose` will print details at each step of what your agent did. In particular, it will show in ASCII art its current location on the grid as well as the one of the Wumpus (which is moving).

The running of your agent follows a general procedure that will be shared for all later practicals:

- The environment generates an observation
- This observation is provided to your agent via the `act` method which chooses an action

- The environment processes your action to generate a reward
- this reward is given to your agent in the `reward` method, in which your agent will learn from the reward

This 4-step process is then repeated several times, either until the end of the game, or until a maximum number of steps (N) has been performed.

Grading

The final performance of your agent will be evaluated by running the following command on a pseudo-random[1] testbed:

```
python main.py --ngames 1000 --niter 100 --batch 200
```

Once you think your implementation is good, create a zip file containing your `agent.py` file and the `metadata` file provided in the template, and upload it to the platform. Your score will be computed and you can compare yourself to the rest of the class using the leaderboard. Your grade for this exercise will be based on this score.

Note: for each game, your agent will be trained over 1000 episodes (with the same setup); but the score will be computed on the last 100 episodes only. Thus, you might want to switch from an exploration mode to an exploitation mode at the 900th episode (or continue training, depending on your approach).

[1] This means that uploading two times the exact same code will generate the exact same score