Project Part 1

Deep Reinforcement Learning

MVA 2022-2023

Denis Duval, Erwan Fagnou, Dorian Gailhard

denis.duval13@gmail.com, erwan.fagnou@gmail.com,
dorian.gailhard@telecom-paris.fr

Introduction

The goal of this project is to solve a given Maze environment with Reinforcement Learning and Deep Reinforcement Learning algorithms. We had to implement one tabular reinforcement learning algorithm and one deep reinforcement learning algorithm in order to solve the environment that was given to us. We could choose among the algorithms that we had seen and implemented in the labs, and adapt them to the given environment.

Environment

The given environment is a Maze in two dimensions. The state of the agent is represented by a tensor of rank 3: the first two dimensions represent the position of the cell and the last dimension informs in a one-hot manner the type of the cell located. More precisely, there are 5 types of cell : Wall, Empty, Player, Goal, Pit. Hence the state is a tensor of shape $M \cdot N \cdot 5$ where M, N are respectively the number of rows and the number of columns of the maze. The values of this tensor can only be $True$ and $False$, with the constraint that a cell has exactly one $True$ value (i.e. one type). Hence, even if the state space is embedded in $\mathbb{R}^{M \cdot N \cdot 5}$, it is finite.
The action space is also finite : the agent can only move in the four classic directions. The reward is 0 for every transition leading to an empty cell, is 1 for every transition leading to the goal and is -1 for every transition leading to a pit. When the agent tries to move on a Wall, it remains on the same cell. When the agent reaches the goal, or a pit, the episode ends.

To explain this choice of representation of the state space, we should say that at every episode the positions of the Goal, Player and Pits are randomly drawn and hence we could not adopt a simple representation of the state of the agent in $\mathbb{R}^2$.

Two levels of difficulty

Finally, we were given two levels of difficulty of the maze. Namely we were given two templates of maze that we will further denote by 'Simple Maze' and 'Bigger Maze'.

Algorithms deployed and results

We have chosen to implement Q-Learning and DQN. In the appendix of this report, you will find in Figure 1 and Figure 2 the best plots obtained for these methods.
Because the performance we obtained on the bigger maze was poor we wanted to implement the other deep reinforcement learning algorithms. The best results we obtained overall on the bigger maze were those obtained with REINFORCE, as depicted in Figure 3 of the Appendix. We can see that the associated policy is almost perfect but there are some cells where the taken action is not optimal and deviate or block the agent from reaching the goal.

Findings and details of implementation

In deep reinforcement learning, here are the parameters we have played with:

- the Q-network. We have tried several architectures for the Q-network. In short, we have tested a simple linear network, a simple convolutional network and a more complex convolutional network. We have designed relatively simple networks because the state space was relatively small. From the experiments we have made, the best results came from the simplest networks, the simple linear network and the the simple convolutional network (that is in fact taken from a lab). The best results, either for DQN or REINFORCE, came from the simple convolutional network.
  We think that the state space is really simple so we don't need complex architectures with a huge number of parameters to understand the maze. Also, the idea of convolutional networks was natural because of the shape of the states of the agent.

- The exploration. This was the main issue in this project. In fact for the agent to converge (in their simplest implementation, i.e. the one of the courses and the labs) they have to find the goal which is only possible by ϵ-greedy exploration. In the simple maze this exploration was quite feasible, leading to a relatively quick convergence although we had to set relatively high values for ϵ in order to explore deep enough.
  For the bigger maze, this exploration is obviously not suited. In fact, this type of exploration leads to very chaotic exploration and the probability that the agent reaches the goal with this exploration is too low. So, as we experimented with DQN, the agent ends doing a suboptimal policy : the two that we have seen the most is the agent going into walls indefinitely or doing cycles between two empty cells. The reason for this is that the reward is very sparse so the agent will never be aware of better rewards.
  Naturally, we first increased the ϵ parameter (in the last runs it was set to 1) and introduced an exponential decay (in the finals runs it was set to 0.9995). For DQN, this increased a lot the performance on the simple maze and increased a bit the performance on the bigger maze.
  The first upgrade that we implemented was a Prioritized Buffer Replay. The idea was to sample the buffer from a distribution that is not uniform, with higher chances to draw transitions with high priority, with the priority defined by the temporal difference of the transition. This didn't improved the performance of DQN on the bigger maze. Plus, the training was much slower compared to uniform replay buffer because of the sampling and the computations of the priorities.

  Another idea to workaround the agent going indefinitely into walls was so give a slightly negative reward to agent each time it goes into a wall. Even if this can be considered as some kind of modification of the environment, we used it to avoid this behaviour of the agent. In fact, this modification doesn't really add a bias to the agent because the optimal policy shouldn't use these kind of actions (i.e. to stay on a cell). This modification led to an increase of performance in DQN.

  The last idea, which could be considered as 'doping the agent', was to guide ourself the agent in its exploration. Intuitively, we want the agent doing "intelligent" exploration i.e. explore rationally, almost as a human. However we didn't find a way to do it without not guiding it too much. So at training, with a probability p_{opt} we executed an optimal trajectory with the agent only 'observing' the transitions but not 'acting'. This sequence of optimal actions was obtained from a 'shortest path' solving algorithm (we used a BFS). We think this helped the agent being aware of the optimal trajectories that exist in the maze. Although, to let the agent discover the suboptimal actions we introduced a decay on the parameter p_{opt} while we also kept high values for ϵ-greedy exploration. This was the upgrade that led to the highest improvement in the mean reward for DQN and REINFORCE.

- The hyperparameters. The tuning of these parameters is in general critic to get the best performance for a given model however in our case this tuning didn't lead to significant changes as the main problem was the exploration.
- An other finding was that in REINFORCE, the mean reward has a high variance even if we can see a linear (positive) trend. The first reflex was to decrease the learning rate but this didn't change anything except slowing the trend. If we increased the learning rate it would only speed up the trend. In both cases the variance in the mean reward was similar so we assumed that this variance was inherent to this agent of reinforcement learning and not linked to the learning rate. Hence we took higher values than for DQN to get the fastest training.

We could also highlight a trick that made the training considerably faster. We added a condition for an episode to end: if the agent has visited a certain cell more than a certain limit (to adjust) then the episode ends. This prevents two issues:

- the agent being stuck on a cell
- the agent doing cycles

In the same vein, we artificially boosted exploration by requiring the agent to explore if it has visited a cell a certain number of times (a lower limit than the one for the episode to end). The idea is to get out of this cycle.