Project DeepRL - Part 1

Clémence Grislain Julien Khlaut Aubin Tchoi

Part 1: Tabular RL - Q-Learning

Small Maze

The hash function and hyperparameters

The state of the maze is an array of size $W * H * n_{\text{cell types}}$ with $n_{\text{cell types}} = 5$: the walls, the void, the pits, the goal and the player. To keep only the useful information, we encode the state in a vector composed of the coordinates of the **player**, of the **goal** and of the **pits**: $(x_s, y_s, x_g, y_g, x_{p_1}, y_{p_1}, \dots, x_{p_n}, y_{p_n})$

We implemented a simple `Qlearning` algorithm and tuned the following parameters:

- γ: the value 0.9 seems to be optimal probably because it allows the algorithm to differ a reward and explore a bit the surroundings while compelling the algorithm to end fast.
- ϵ: represents the exploitation-exploration trade-off. On one hand if ϵ is too high ≈ 0.8 the agent reaches quickly the goal the first time ($\approx$ 250 episodes) but struggles learning from there because it can only explore and not exploit. On the other hand, with smaller $\epsilon \approx 0.3$ then the model succeeds in learning but takes a longer time to reach the goal ($\approx$ 500 episodes).
- α: the agent performs the best in the interval [0.05, 0.1].

During the experiment, we used adaptative strategies for ϵ and α: instead of having a fixed value we initially set $\epsilon = 0.9$ and decrease it by a factor $\lambda_\epsilon = 0.7$ every $N_\epsilon = 250$ episodes. We applied the same strategy for α, initially set to $\alpha = 0.05$ and multiply it by a factor $\lambda_\alpha = 1.5$ every $N_\alpha = 500$ episodes. That way we can let our algorithm explore and quickly reach the goal and then exploit as much as possible.

Results

As shown in the Figure 1 a), the average evaluation return of `Qlearning` successfully converges towards 1 for $\approx$ 1300 episodes. Finally, the agent trained on 2000 episodes with parameters ϵ, α and γ as described above, achieves **0.89** $\pm$ **0.2** of average return (95% confidence interval on 100 evaluation runs).

Big Maze

On the big maze, using the algorithm described above does not provide results whatever hyperparameters are used. In `Qlearning` algorithm, the agent leans a path for every possible hashed state. Hence, for every possible pits configurations. Yet, the number of potential pits from the small maze to the big maze went from 4 to 15 which multiplies the number of state by $15! \approx 10^{12}$.

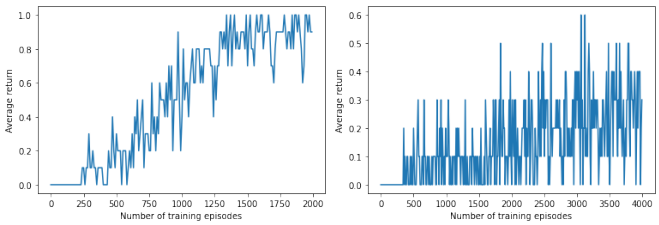
To solve this issue we changed the hash function to use the position of all the potential pits $\{p_i\}_N$ instead of the position of the pits actually present in the run. The new hash function is $(x_s, y_s, x_g, y_g, x_{p_1}, y_{p_1}, \dots, x_{p_n}, y_{p_N})$.

Remark: the agent cannot distinguish the real pits from the potential ones, and such configuration can lead to dead ends. Yet, we expect the agent to learn to avoid potential pits with a smaller aversion than previously, as walking on a potential pit will only lead to a reward of -1 with probability strictly inferior to 1.

Results

In practice during the training, the agent does not reach a mean evaluation return higher than 0.3 (on 10 runs evaluation). This phenomenon is caused by the maximum number of steps being too small. When the agent S and the goal G are too further apart, the agent ends up with no reward if it did not reach a pit nor the goal. In fact, after training, only 10% of the states had a non null Q-value which means that most of the time in evaluation our agent had no incentive on the action to take.

Increasing the number of steps the agent can learn from partially solves this problem. Indeed if we increase the max_step from 1000 to 10000 we can reach an average reward of 0.6 on 10 evaluation runs during the training, as shown in Figure 1 b), and results in 20% of states with non null Q-value. Finally, the agent can solve the maze $\approx 60\%$ of the time but fails for $\approx 40\%$ of the mazes, which happens to be configurations where S and G are very far away.



(a) Small maze (b) Big Maze

Figure 1. Convergence of `Qlearning` algorithm (10 evaluation runs every 10 episodes)

Part 2: Deep RL - DQN

Small Maze

We chose to implement a `DQN` agent with a replay buffer. The model acts accordingly to an ϵ-greedy policy during training, and adopts the greedy policy with regard to its estimation of the Q-function when evaluated. Our implementation involves two models, an online model updated using the MSE as its loss and `Adam` optimizer, and a target network that is updated at each step by a convex combination of its parameters and the online network's parameters. The agent therefore holds many hyperparameters, among which the discount factor γ, ϵ, α (update of the target network), the size of the replay buffer, the learning rate and the architecture of the networks.

Fine-tuning of the hyperparameters

To fine-tune γ, ϵ and the size of the replay buffer we tracked the learning process throughout each training sample. We first observed that a value of $\epsilon = 0.7$ was fit for exploration on the smaller maze because reaching the goal the first time is not an issue there, and setting a value too high would hinder the learning process. We found that having a small capacity for the replay buffer decreases performances as it leads to drawing correlated samples, which brings instabilities in the training.

Results

We tried out various architectures, starting from a simple MLP to small convolutional architectures. The best results are obtained using architecture with two convolutional layers of sizes 16 and 32 (with ReLU activation) and one final dense layer of size 64 or 128, see Table 1 from the annex A. The `DQN` algorithm, trained on 1000 episodes with the parameters, fixed $\epsilon = 0.7$, $\gamma = 0.9$, $\alpha = 0.9$, learning rate of 3e-4, buffer capacity of 2500 and batch size of 32, achieves an average reward of $\mathbf{1.}\pm\mathbf{0.0}$ (95% confidence interval on 100 evaluation runs). The training took $\approx 6m$ without the intermediate evaluation.

Remark: we tested the robustness of this result for the architecture [16,32]+[128] and found that for 10 different attempts, the average reward on 100 evaluations runs stays equal to $\mathbf{1.}\pm\mathbf{0.0}$.

Big Maze

Similarly to what we encountered when using Q-learning our method did not generalize immediately to the bigger maze. Our first observation was that our agent seldom reaches the goal and therefore has nothing to learn from as it then does not receive a reward. To compensate for this issue we extended the maximum size of an episode and increased the value of ϵ to 0.8 to promote exploration. Increasing the capacity of the replay buffer also proved to be useful, which is expected because there are significantly more possible states on the bigger maze, and therefore if we keep the same capacity we might end up drawing states that are highly correlated. To make up for the increased lengths of the paths linking each starting point to each goal, we increased the value of γ, which causes our agent to account for more distant future observations.

The main drawback we faced on the bigger maze lied in the increased time required to train the model. One improvement we brought comes from the observation that the agent often gets stuck in evaluation. Indeed, our evaluation policy does not contain any randomness, and therefore if the agent loops on its position, it cannot get out of this loop. The most common case were loops of length 1 where the agent would run into a wall, and we opted to end the episode when this behavior was detected.

Learning from increasingly complex mazes

Since our model was efficient on the small maze but did not generalize to the bigger maze, we thought of leveraging the amount of learning performed on the smaller, easier maze to transition onto the bigger maze. To push the idea even further, we designed a series of 7 mazes of increasing complexity, starting from a simple maze that does not have any wall, and ending with the big maze. We then train iteratively an agent on these mazes by transferring the `LearnerState` from one run onto the next. This method has the benefit of allowing for more efficient training, but it is sensible to the choice of the intermediate mazes. The first one contains the same potential starting points and goals than the big maze but no wall. We then start adding the walls progressively in the next two mazes, and then start adding more and more pits using the maze parameter P.

Unfortunately, our estimation of the runtime required to train the model on all the mazes (more than 10 hours) largely exceeded the limit enforced by Google Colab and we were therefore unable to fully train the model on the big maze.

A. Annex

Conv sizes	MLP size	Mean return
None	32	0.4 ± 0.1
None	64	0.73 ± 0.09
None	128	0.61 ± 0.09
[4, 8]	32	0.98 ± 0.03
[4, 8]	64	0.91 ± 0.06
[4, 8]	128	0.85 ± 0.07
[16, 32]	64	1.0 ± 0.0
[16, 32]	128	1.0 ± 0.0

Table 1. Mean return and 95% c.i on 100 evaluations runs for different MLP and convolutional architectures trained of 1000 episodes with the parameters $\epsilon = 0.7$, $\gamma = 0.9$, $\alpha = 0.9$, learning rate of 3e-4, buffer capacity of 2500 and batch size of 32