Reinforcement Learning
TD 1 - MDP

Fabien Pesquerel
fabien.pesquerel@inria.fr

Odalric-Ambrym Maillard
odalric.maillard@inria.fr

Patrick Saux
patrick.saux@inria.fr

March 16, 2021

1 Questions from the audience

You can write below the questions that were asked during the session and see later if you can remember/re-derive the answers.

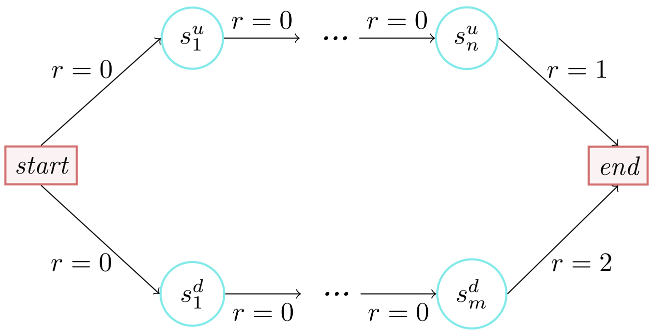
2 About the course

Exercice 1. *Given a MDP with bounded rewards, $M = (S, A, p, r)$, and a deterministic policy π,*

1. *Recall the Bellman operator associated to the policy π and discounted reward γ,*
2. *Prove that the Bellman operator is a γ-contraction for the infinity norm.*

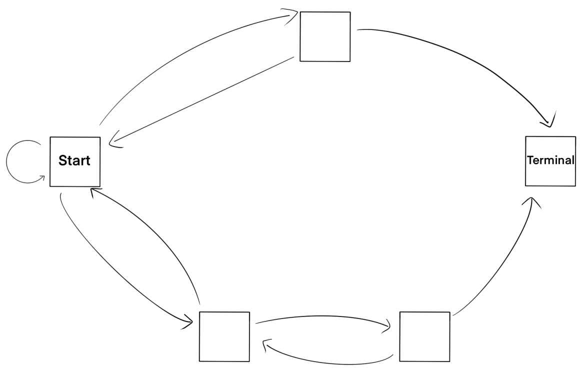
3 Exercises to build the intuition

Exercice 2 (On the influence of the discount factor)**.** *We consider this simple MDP in which all the transitions are deterministic.*



1. *Compute $Q(s, up)$ and $Q(s, down)$ as functions of γ (the discount factor), n (length of the upper chain, i.e., the number of states after the initial state when choosing up from that state) and m (length of the lower chain, i.e., the number of states after the initial state when choosing down from that state).*
2. *Compute the optimal policy π_* as a function of γ, n and m.*

Exercice 3 (On the influence of the reward function)**.** *We consider this simple maze.*



1. *Knowing that all transitions are deterministic, what are some strategies that allows to escape the maze?*
2. *Which rewards and function of rewards[1] would you choose to compute an escape policy?*
3. *Which rewards and function of rewards would you choose to compute an escape policy that uses the minimal number of moves?*

[1]Discounted, average, ...

4 Apply the theory

Students can hope to come back to this section later in the course and test what they will learn (algorithms, methods, heuristics). In particular, Deep RL ideas might be tested using the following exercise. Students are assumed to be familiar with *python 3.X* and usual scientific libraries. In particular, it is recommended to install *numpy*, *scipy*[2], *gym* and *pyTorch* (that will be used later in the course).

Exercice 4 (Gym & Frozen Lake)**.** *Frozen Lake is a gym environment that we will use to implement some algorithms that were learned during the first class. In the following, we will use a discount factor of $\gamma = 0.9$.*

1. *To check that everything is installed on your computer/environment run the small script* ***check_env.py****. It will try to import the necessary libraries and print current versions of them. You can use those versions to help you with any troubleshooting.*

2. *To better grasp the Frozen Lake environment, read, understand and run* ***discover.py*** *(until the stop mark). Describe the MDP associated to this environment and formalize mathematically the goal of an agent.*

3. *Using the aforementioned environment, get a Monte-Carlo estimation of the value function of a simple deterministic policy at s_0, the initial state. For instance, this simple policy could be to always going to the right. This is the implemented example in* ***discover.py****.*

4. *In the case of finite MDPs, the value function V^π of a policy π is a vector in $\mathbb{R}^{|S|}$ (Tabular case). Write V^π as the solution of a the product of a matrix with a vector (both known once π is). Write a function that takes a deterministic policy π as an input and outputs its value function V^π. Compare the result of this question with the Monte-Carlo estimation of the previous question and check for consistency.*

5. *Using the contraction property of the Bellman operator, implement a function that iteratively apply the Bellman operator to compute the value function V^π of a policy π. What could be a good stopping criterion for your algorithm?*

6. *Using the contraction property of the optimal Bellman operator, implement a function that iteratively apply the optimal Bellman operator to compute the optimal value function V^*.*

7. *Compute an optimal policy for this environment using Value Iteration (see appendix).*

8. *Compute an optimal policy for this environment using Policy Iteration (see appendix).*

9. *Compute an optimal policy for this environment using Q-learning (see appendix). Note the change of setting: you need to simulate several episodes of Frozen Lake and learn from observed transitions only (for instance you may not assume the transition probabilities are known). Feel free to play with the number of episodes, learning rate and exploration parameter ε. You may set the is_slippery argument to False when declaring the environment to have deterministic transitions and help the algorithm learn faster.*

10. *Compare quantitatively the three methods.*

[2]Those that are looking for performance might want to take a look at sparse representation of matrices (*scipy.sparse.csr_matrix*) and computation's techniques with arrays (BLAS and LAPACK are used as routine in *scipy*).

A Algorithms

```
Algorithm: Value Iteration
Input: MDP transition kernel $\mathbf{P}(s' \mid s,a)$ and average reward $\mu(s,a)$, discount factor $\gamma$,
  number of iterations $N$.
Initialisation: $v_0 \in \mathbb{R}^{|\mathcal{S}|}$, $n = 0$.
while $n < N$ do
    for $s \in \mathcal{S}$ do
        $v(s) \leftarrow \max_{a\in\mathcal{A}} \mu(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) v(s')$;
    end
    $n \leftarrow n+1$;
end
Return: $\pi \in \arg\max_{a\in\mathcal{A}} \mu(\cdot,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid \cdot,a) v(s')$.
```

```
Algorithm: Policy Iteration
Input: MDP transition kernel $\mathbf{P}(s' \mid s,a)$ and average reward $\mu(s,a)$, discount factor $\gamma$.
Initialisation: $\pi \in \mathcal{A}^{\mathcal{S}}$.
while policy not stable do
    for $s \in \mathcal{S}$ do
        $v_\pi(s) \leftarrow (I - \gamma \mathbf{P}_\pi)^{-1} \mu_\pi(s)$;                         // policy evaluation
        $\pi(s) \leftarrow \arg\max_{a\in\mathcal{A}} \mu(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s,a) v(s')$;   // policy improvement
    end
end
Return: $\pi$.
```

```
Algorithm: Q-Learning (episodic, $\varepsilon$ greedy)
Input: Discount factor $\gamma$, learning rates $(\alpha_t)_{t\in\mathbb{N}}$, number of episodes $T$, $\varepsilon > 0$.
Initialisation: $Q \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, $t = 0$.
while $t < T$ do
    $h \leftarrow 0$;
    Start episode $t$ in state $s_h$ and action $a_h$;
    while episode not terminated do
        Take action $a_h$ in state $s_h$, observe reward $r_h$ and move to state $s_{h+1}$;
        $Q(s_h,a_h) \leftarrow Q(s_h,a_h) + \alpha_t\big(r_h + \gamma \max_{b\in\mathcal{A}} Q(s_{h+1},b) - Q(s_h,a_h)\big)$;
        $a_{h+1} \in \arg\max_{b\in\mathcal{A}} Q(s_{h+1},b)$ with probability $1-\varepsilon$ else $a_{h+1} \sim \mathrm{Unif}(\mathcal{A})$;
        Check if episode $t$ is terminated;
    end
    $t \leftarrow t+1$;
end
Return: $\pi(s) \in \arg\max_{a\in\mathcal{A}} Q(s,a)$
```