Reinforcement Learning
TD 1 - MDP

Fabien Pesquerel
fabien.pesquerel@inria.fr

Odalric-Ambrym Maillard
odalric.maillard@inria.fr

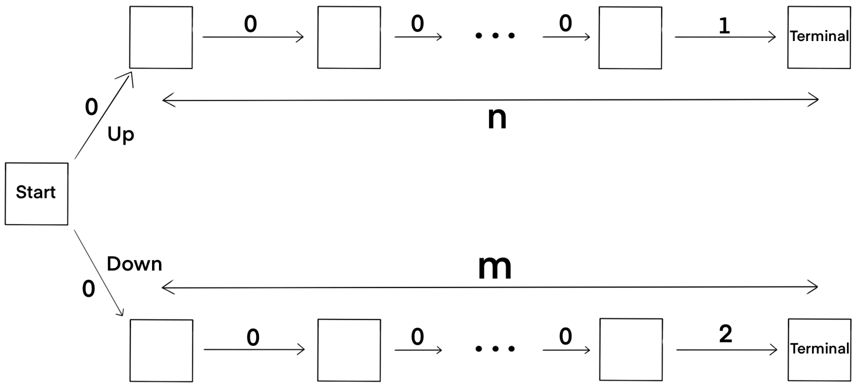
January 6, 2020

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 Exercises to build the intuition

Exercice 1 (On the influence of the discount factor). *We consider this simple MDP in which all the transitions are deterministics.*



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.*

Answer - Exercice 1

- 1. There are only two policies on this deterministic MDP which can be denoted as *up* and *down.* This is because there is only one available action in each state except for the first one. Because of that, the expected values that we want to compute are just real sums. By definition of the state-action value function, one can write:

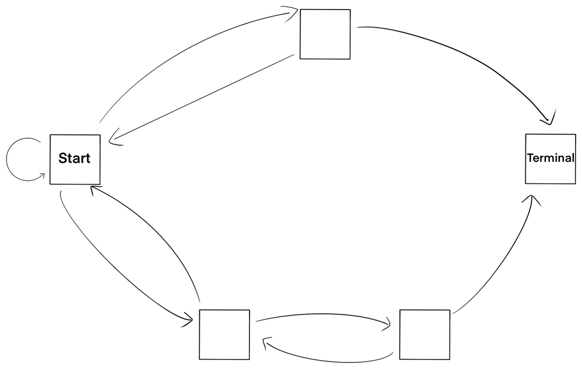
$$Q(start, up) = \mathbb{E}_{\pi_{up},MDP}\left(\sum_{t=0}^{n}\gamma^t r_t | s_0 = up, a_0 = up\right) = \sum_{t=0}^{n}\gamma^t r_t = \gamma^n$$
$$Q(start, down) = \mathbb{E}_{\pi_{down},MDP}\left(\sum_{t=0}^{n}\gamma^t r_t | s_0 = up, a_0 = down\right) = \sum_{t=0}^{n}\gamma^t r_t = 2\gamma^m$$

- 2. Since there are only two policies, one only have to compare the two previous computed quantities:

$$\frac{Q(start, up)}{Q(start, down)} = \frac{1}{2}\gamma^{n-m}.$$

  Therefore, $\pi_* = \pi_{up}$ if $\gamma^{n-m} > 2$ and $\pi_* = \pi_{down}$ if $\gamma^{n-m} < 2$. The case $\gamma^{n-m} = 2$ can be broken arbitrarily.

Exercice 2 (On the influence of the reward function). *We consider this simple maze.*



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

Answer - Exercice 2

- 1. We are considering deterministic policies. The space of all *valid* policies (the one that allows to reach the terminal) is made of all the policies that create a path from *start* to *terminal.* One can map *Start* to either *up* or *down.* If *up* has been chosen, one can prescribe whatever we want for the lower chain and necessarily choose the *right* action after the *up* transition. If *down* has been chosen as the first action, one can prescribe whatever we want for the upper chain and necessarily choose the *right* actions after the *up* transition in all subsequent states of the lower chain.
- 2. No strictly positive reward for every transitions except the ones pointing to the terminal is a necessary but not sufficient condition. For instance, if all rewards are 0s, we may not learn to escape the maze. Instead, the difference between the rewards of transitions to the terminal and other rewards should be strictly positive.
- 3. We may do one of the two things : modify the reward function by using a discount factor or modify the rewards of the MDP. If all rewards are 0s except the ones associated to transitions to the terminal that we set equal to 1, then the cumulative reward computed using a discount factor of γ is γ^T. T is the time spent within the maze before escaping it using a policy. The larger the time, the smaller the cumulative reward. Hence, we will compute a policy escaping with a minimal number of moves. One can also set all rewards of non-terminal transitions to -1 and set the rewards terminal transitions to 0. In this case, the cumulative reward is $-T$ where T is the same quantity as before. Once again, the smaller the time spent in the maze, the larger the cumulative reward.

Exercice 3 (Monte Carlo estimation and value functions)**.** *Let $W : \Omega \to \mathbb{R}$ be a L_1 random variable. We denote by $(w_i)_{i\geq 1}$ a sequence of i.i.d samples drawn from W. We denote u_k the empirical means computed using the first k samples of the sequence:*

$$u_k = \frac{1}{k}\sum_{i=1}^{k} w_i.$$

1. *Write u_{k+1} as a function of k, u_k and w_{k+1}.*
2. *With a Monte Carlo mindset, which function are we stochastically minimizing?*

Answer - Exercice 3

- 1. $u_{k+1} = u_k - \frac{1}{k+1}\left(u_k - w_{k+1}\right)$
- 2. The keyword is Stochastic Gradient Descent. The expected value of a real random variable is the minimizer of the dispersion function:

  $$\mathbb{E}\left(W\right) = argmin_x \frac{1}{2}\mathbb{E}\left(x - W\right)^2.$$

  Writing $j(x,w) = \frac{1}{2}\left(x-w\right)^2$:

  $$\mathbb{E}\left(W\right) = argmin_x \mathbb{E}\left(j\left(x, W\right)\right).$$

  Therefore, the online update of the mean can be written as a stochastic gradient descent on a convex function:

  $$\begin{aligned} u_{k+1} &= u_k - \frac{1}{k+1}\left(u_k - w_{k+1}\right) \\ &= u_k - \frac{1}{k+1}\nabla_x j\left(u_k, w_{k+1}\right) \\ &= u_k - \epsilon_k \nabla_x j\left(u_k, w_{k+1}\right) \end{aligned}$$

  and we can chose ϵ_k to be any sequence that satistfy the Robbins–Monro conditions.

Take home message: Computing an expected value can be seen as a stochastic minimization task.

Exercice 4 (Fixed point methods and expectancy)**.** *We throw a die with n faces which are numerated from 1 to n.*

1. *On average, how many times must this die be rolled until a given figure turns up?*
2. *On average, how many times must this die be rolled until there are two rolls in a row that differ by 1.*

Answer - Exercice 4

- 1. We show that this quantity can be computed as a fixed point of a contraction mapping. Let T denote the stopping time that count how long we wait before seeing the chosen figure, say x, for the first time. Denote by $(X_k)_k$ the sequence of *i.i.d.* figures.

  $$\begin{aligned} \mathbb{E}\left(T\right) &= \mathbb{E}\left(T|X_1 = x\right)\mathbb{P}\left(X_1 = x\right) + \mathbb{E}\left(T|X_1 \neq x\right)\mathbb{P}\left(X_1 \neq x\right) \\ &= 1 \times \frac{1}{n} + \mathbb{E}\left(T|T \geq 1\right) \times \frac{n-1}{n} \\ &= 1 \times \frac{1}{n} + \left(1 + \mathbb{E}\left(T\right)\right) \times \frac{n-1}{n} \\ &= 1 + \frac{n-1}{n}\mathbb{E}\left(T\right) \end{aligned}$$

  And we get the classical result: $\mathbb{E}\left(T\right) = n$.

- 2. Again, we link this quantity to the computation of a contraction mapping. Let's denote by E the quantity that we want to compute. Let E_k denotes the average time we have to wait before the condition is satisfied after getting figure k. Then,

  $$E = 1 + \frac{1}{n}\sum_{k=1}^{n} E_k.$$

  Writing $X = \left(E_1, \cdots, E_k\right)^T$, one can compute X as the fixed point of a linear function in a similar fashion than the first question. Denoting $\Omega = (1 \cdots 1)$ and H is the matrix full one 1s, we have:

$$X = \Omega + \frac{1}{m}\left(H - \begin{pmatrix} 0 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 0 \end{pmatrix}\right) X$$

Take home message: Computing an expected value can be seen as computing the fixed point of contraction mapping.

3 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*[1], *gym* and *pyTorch* (that will be used later in the course).

Exercice 5 (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.*
8. *Compute an optimal policy for this environment using Policy Iteration.*
9. *Compute an optimal policy for this environment using Q-learning.*
10. *Compare quantitatively the three methods.*

[1]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*).