Reinforcement Learning
TD 1 - MDP

Fabien Pesquerel
fabien.pesquerel@inria.fr

Odalric-Ambrym Maillard
odalric.maillard@inria.fr

January 11, 2022

Students can hope to come back to this practical session 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 1 (Gym & Frozen Lake). *Frozen Lake*[2] *is a gym environment that we will use to implement some algorithms that were learned during the lessons. 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****. 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 at* s_0 *of a simple deterministic policy (*s_0 *is the initial state). For instance, this simple policy could be to always choose the* `RIGHT` *action. This is the implemented example in* ***hint_question3.py****.*
4. *(****Expected value method****) Write a function that takes a deterministic policy* π *as an input and outputs its value function* V^π*. The function should use Monte-Carlo estimation to compute the value function* V^π*.*
5. *(****Linear system method****) If* $M = (S, A, p, r)$ *be a finite MDP, then 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.*
6. *(****Bellman operator method****) Knowing the contraction/convergence property of the Bellman operator, implement a function that iteratively applies the Bellman operator to compute the value function* V^π *of a policy* π*. What could be a good stopping criterion for your algorithm?*
7. *Knowing the contraction/convergence property of the* ***optimal Bellman operator****, implement a function that iteratively applies the optimal Bellman operator to compute the optimal value function* V^**.*

[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*).

[2]Warning: The documentation is not up to date with the git repository. To import the Frozen Lake environment, you should `import FrozenLake-v1` rather than `import FrozenLake-v0` (which won't work).

8. *Compute an (approximation of an) optimal policy for this environment using* ***Value Iteration*** *(use the previous question).*

9. *Compute an (approximation of an) optimal policy for this environment using* ***Policy Iteration****.*

10. *Compare the two methods.*

11. *Using the contraction property of the* ***optimal Bellman operator****, implement a function that iteratively applies the optimal Bellman operator to compute the optimal value function Q^*.*

12. *Using the previous question, write a function that computes and an optimal policy from it.*

13. *Render a trajectory from start to finish and print the discounted cumulated reward associated to it.*

Exercice 2 (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

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:

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

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

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

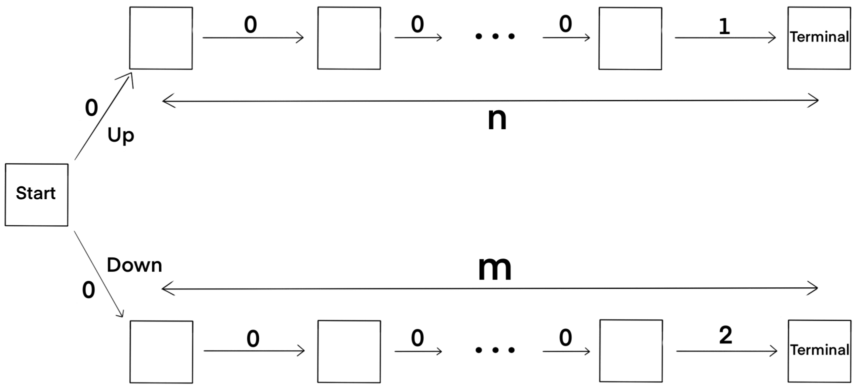
Therefore, the online update of the mean can be written as a stochastic gradient descent on a convex function:

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

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 3 (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

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.