Reinforcement Learning
Homework 1

Amine KHELDOUNI

November, 12^{th} 2018

1 Dynamic Programming

In this section, we consider the Markov Decision Process with stationary reward and transition model illustrated below.

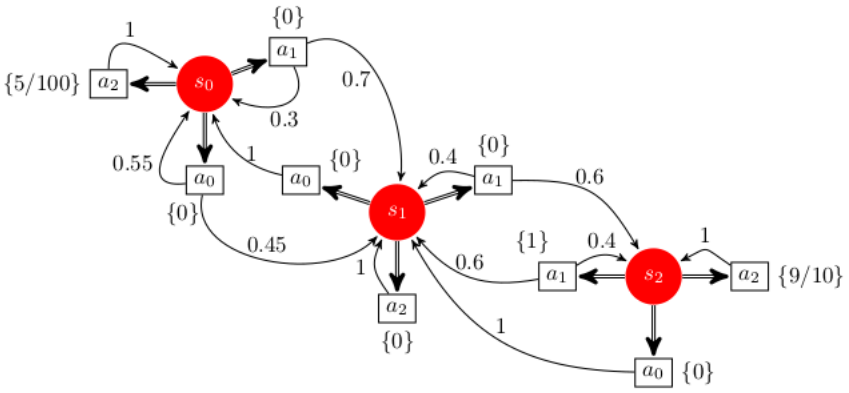

FIGURE 1 – Symbolic representation of a three-state MDP

Q1 : To implement the discrete MDP model, we construct transition matrices and a reward matrix which follows the Markov decision process rules (cf. 1). On the one hand, constructing the transition matrices by action gives the probability that we end up in state s_j given the current state s_i and the applied action a_k. Therefore, we have three transition matrices (one for each action) :

$$T_{a_0} = \begin{pmatrix} 0.55 & 0.45 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad T_{a_1} = \begin{pmatrix} 0.3 & 0.7 & 0 \\ 0 & 0.4 & 0.6 \\ 0 & 0.6 & 0.4 \end{pmatrix} \quad T_{a_2} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

On the other hand, we encoded the reward matrix as $R = (r_{ij})_{i,j}$ being the reward of action i when we apply it from state s_j. Therefore, following the MDP scheme above :

$$R = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0.05 & 0 & 0.9 \end{pmatrix}$$

For such a MDP, it is indeed simple to guess the optimal policy π^*. Since we want to converge towards the stationary point with highest reward (s_2, a_2), we know we want to make the transition from s_0 to s_1 by applying action a_1 and from s_1 to s_2 by applying a_1 again. Therefore, the optimal policy π^* is $\pi^* = (a_1, a_1, a_2)$.

Q2 : Implementing the value iteration (VI) using a discount factor $\gamma = 0.95$, we can identify a 0.01-optimal policy with an optimal value (computed with a policy evaluation) $v^* = (15.2, 16.36, 17.81)$ and a best policy $\pi^* = (a_1, a_1, a_2)$ as expected. We display below the convergence pattern of our VI algorithm, plotting the $(\|v^{k+1} - v^k\|_\infty)_k$ with k the iteration counter and we verify that the stopping criterion (implied by Puterman) is verified (as $\frac{2\epsilon\gamma}{1-\gamma} \approx 0.34$). The VI algorithm converges and we compute

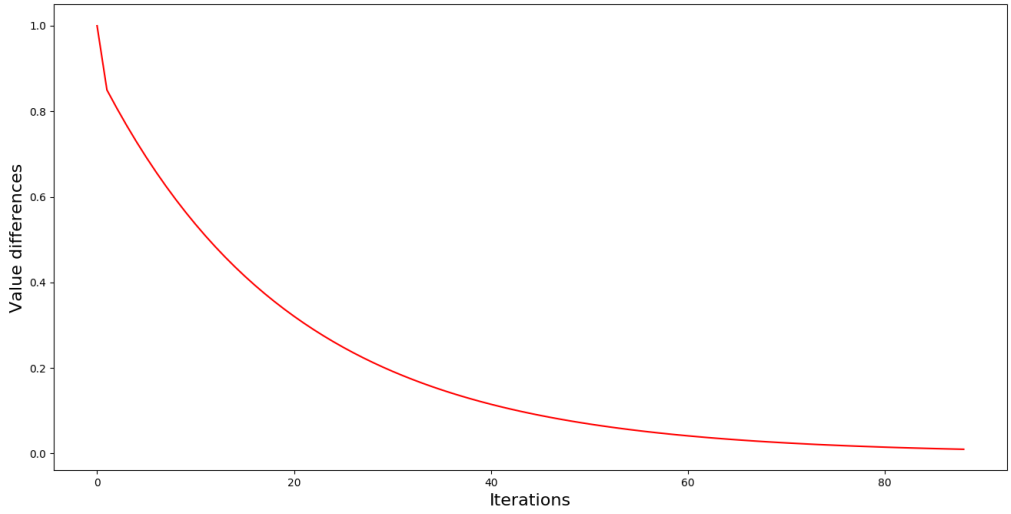

FIGURE 2 – Convergence of VI algorithm on a simple MDP

the optimal value using a policy evaluation. Recollect that :

$$v^* = (15.2, 16.36, 17.81)$$

Q3 : As in the previous algorithm, we use the Bellman dynamic programming equation to implement this time a policy iteration (PI) algorithm. We notice that the exact policy iteration converges well towards the same optimal policy π^*. However, the policy iteration algorithm is noticeably faster that the value iteration in terms of convergence speed since the algorithm finds the optimal policy in 3 iterations instead of 89 iterations.

We deduce that value iteration is simpler but is computationally heavier than policy iteration. In value iteration, every iteration updates both the values and implicitly the policy and we don't keep track of the policy but we recover the optimal policy by taking the maximum over the actions whereas the policy iteration performs several passes that update utilities with fixed policy, then chooses a better policy until convergence.

2 Reinforcement Learning

Q4 : We consider in this section the grid world environment shown in the following figure. Only on this part, we consider the deterministic policy consisting in selecting the action *right* when available and *up* otherwise. Then we want to compute an empirical value function based on Monte Carlo estimations.

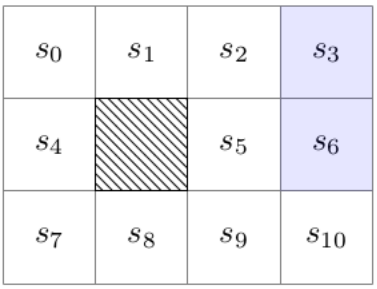

FIGURE 3 – Simple Grid World with zero-based state enumeration

Simulating the starting state, we compute the starting state distribution

$$\mu_0 = (0.142, 0.091, 0.092, 0.042, 0.09, 0.092, 0.089, 0.091, 0.0897, 0.09, 0.0898)$$

and we build an estimator based on the empirical average :

$$V_n(s) = \frac{1}{N(s)} \sum_{k=1}^{N(s)} \left[\sum_{t=1}^{T_{max}} {}^{t-1} r_t^{(k)} \right]$$

Plotting the iterations $n \to J_n - J^\pi$ we demonstrate the convergence of such estimator :

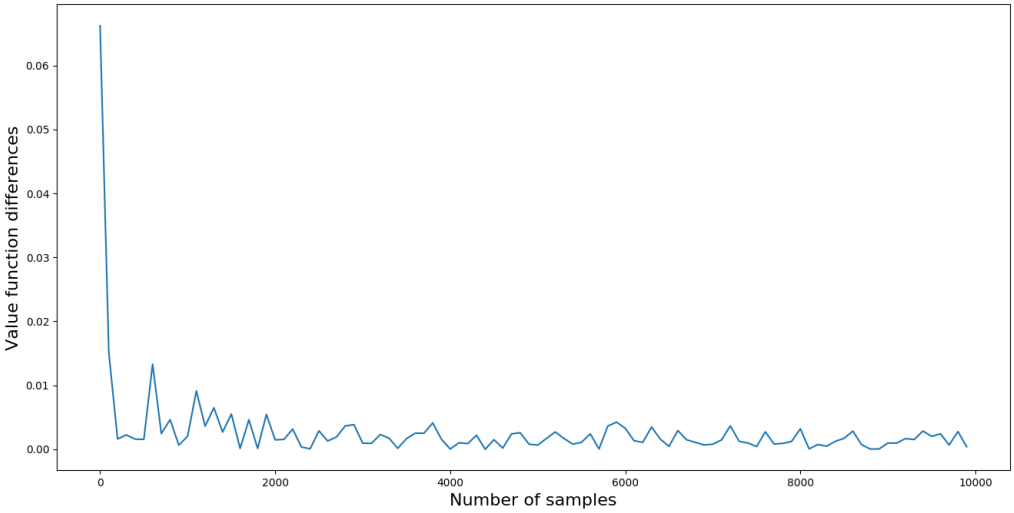

FIGURE 4 – Convergence of Monte Carlo empirical estimator of value function

Q5 : We implement the Q-learning algorithm by simulating trajectories (or episodes) and updating the Q-value along the way using the following equation :

$$Q_{t+1}(x_t, a_t) = (1 - \alpha_{N(x_t,a_t)}(x_t, a_t))Q_t(x_t, a_t) + \alpha_{N(x_t,a_t)}(x_t, a_t)\left[r_t + \gamma \max_{b\in\mathcal{A}} Q_t(x_{t+1}, b)\right]$$

First and foremost, we choose the learning rate to be variable and depending on $N(x_t, a_t)1$ at step t, so that it satisfies the usual stochastic approximation requirements :

$$\sum_i \alpha_i(x, a) = +\infty \quad \text{and} \quad \sum_i \alpha_i(x, a) < +\infty$$

We took

$$\alpha_{N(x_t,a_t)}(x_t, a_t) = \frac{1}{(1 + N(x_t, a_t))^{0.8}}$$

Second, we chose and ϵ for our ϵ-greedy policy to be small enough to not induce too much randomness during action choosing step. After some experiments, we set $\epsilon = 0.001$ since this value explores the possible action set during the episodes and does not induce too much randomness after building an accurate Q-value map. Finally, the T_{max} was taken equal to 20 time steps since the considered grid world does not have a lot of possible states (only 11).

To illustrate the convergence of our Q-Learning algorithm, we illustrate the performance by plotting for each episode n, the performance over all the other state $\|v^* - v^{\pi_n}\|_\infty$ and the the reward cumulated over the episode.

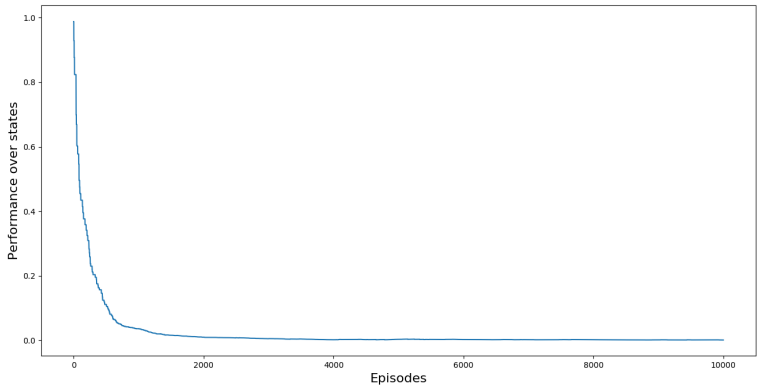

FIGURE 5 – Convergence of the Q-Learning algorithm over the episodes

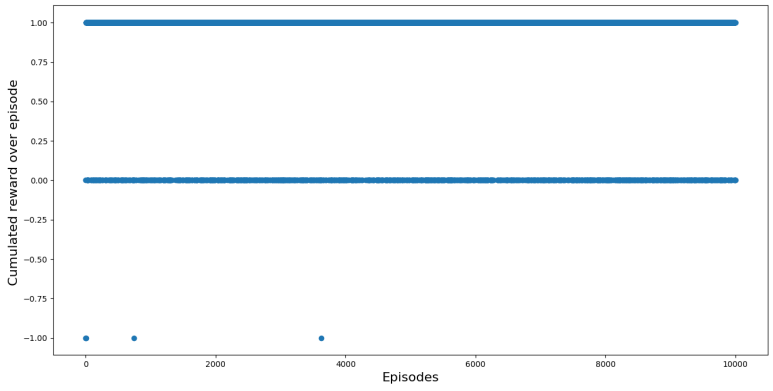

FIGURE 6 – Cumulated reward at the end of each episode in the Q-Learning algorithm

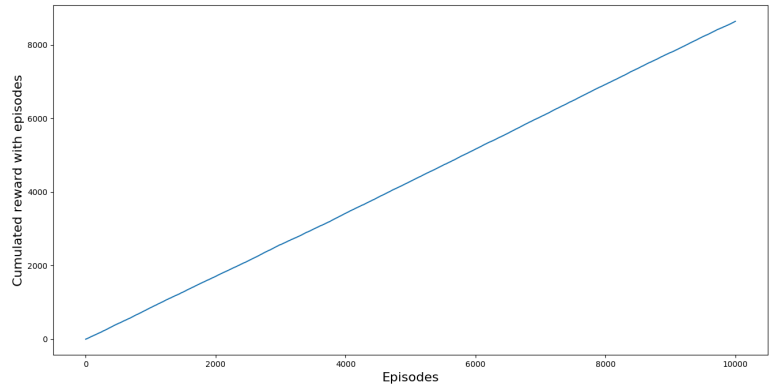

FIGURE 7 – Cumulated reward over the past episodes in the Q-Learning algorithm

We notice that the algorithm fails only few times by getting a negative reward but learns with the episodes the Q-Learning map and the optimal policy choice.

In the following figure, we adapted the *render_q* function to display a 0 reward triangle even when the action is not possible, this helped us to avoid problems with numbering the possible actions for each state (we couldn't build a matrix which displays correctly the Q-values since states may not have the same number of possible actions). Therefore, our grid below validates the Q-Learning algorithm which learns when it is good or bad to perform a certain action given the current state.

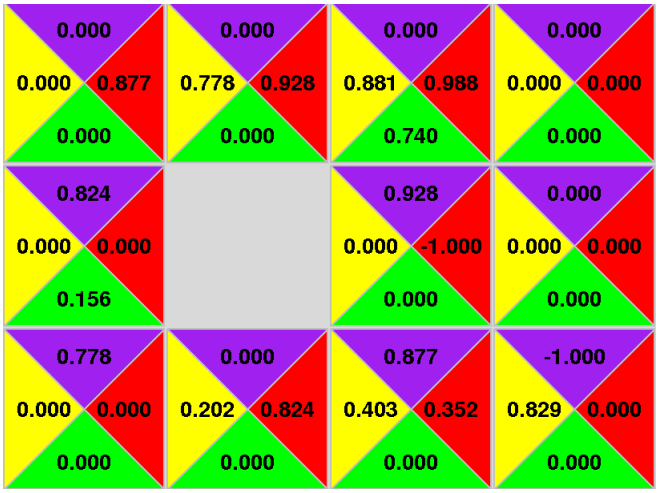

FIGURE 8 – Grid World environment with Q-value rendering

Q6 : The optimal policy of a MDP is not affected by the change of the initial distribution μ_0. This is linked to the Markov property of environment : Markov property states that the history of previous states and actions leading to state s does not affect $R(s)$ and $\mathbb{P}(s'|s,a)$. So in any state s, the optimal policy for that state can only consider the reward $R(s,a)$ and the transition probability for any action a, without considering how it has reached s.