

Dynamic Programming and Reinforcement Learning

Lecturer: Matteo Pirodda

v0.2 (October 30, 2018)

Report Deadline: November 12, h18:00

Your report should be *short* and the code is *not* optional! Please be as clear and concise as possible, this will speed up the correction process!

The solution (single archive with name *lastname_firstname_TP1.zip*) should be uploaded to Dropbox using the following link. *Pay attention to the name of the file!*

⚠ Note that the submission website (i.e., Dropbox file request) will close automatically at the prescribed hour, please submit on time. We will not accept a late submission via email.

We provide support for Matlab and Python but you can solve the homework using any language. If you use a notebook, we kindly ask you to generate a PDF out of it.

Note: if you need help or clarifications use Piazza!

1 Dynamic Programming

The following simple example illustrates the basic components of a Markov Decision Process (MDP) with stationary reward and transition model. Fig. 1 illustrates a simple three-state MDP. The elements in braces denote the rewards, while the weights on the arcs are the transition probabilities. For example, action a_0 in s_0 is such that

$$p(s_0|s_0, a_0) = 0.55, \quad p(s_1|s_0, a_0) = 0.45 \quad \text{and} \quad r(s_0, a_0) = 0$$

Assume a discount factor $\gamma = 0.95$.

- Q1: Implement the discrete MDP model. In this simple MDP it is simple to guess the optimal policy π^* . What is it?
- Q2: Implement and run value iteration (VI) in order to identify a 0.01-optimal policy. Recall that the stopping criterion (theory slides) is to stop when

$$\|v^{k+1} - v^k\|_\infty < \epsilon.$$

which implies that [Puterman, 1994, Th. 6.3.1]

$$\|v^{\pi_{k+1}^+} - v^*\| < \frac{2\epsilon\gamma}{1-\gamma},$$

where π_{k+1}^+ is the greedy policy w.r.t. v^{k+1} .

Plot $\|v^k - v^*\|_\infty$ as a function of iteration k . Implement policy evaluation to compute v^* , i.e. the value function of the optimal policy. Guess (compute) the optimal policy by looking at Fig. 1.

- Q3: implement **exact** policy iteration (PI) with initial policy $\pi_0 = [a_0, a_0, a_0]$. Compare the speed of convergence w.r.t. VI and discuss the relative merits of the two approaches.

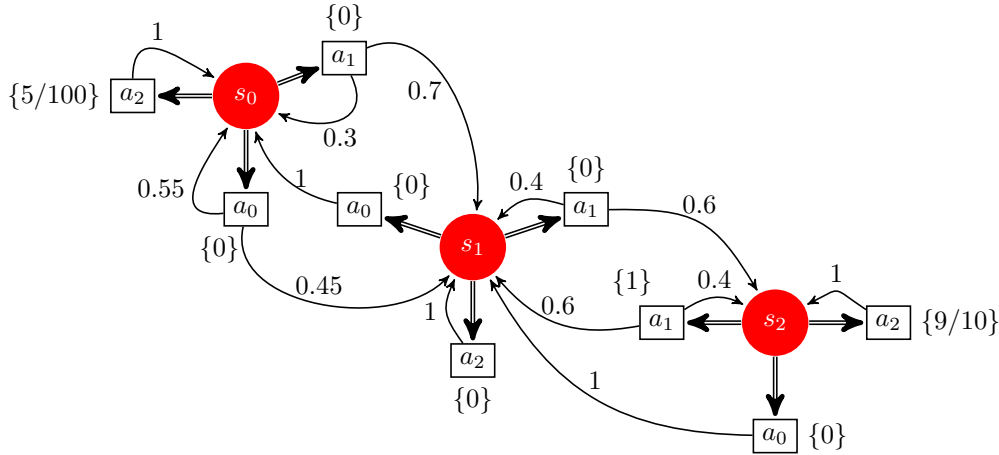


Figure 1: [Exercise 1] Symbolic representation of a three-state Markov decision process.

2 Reinforcement Learning

In the previous exercise, we have investigated methods that require the knowledge of the MDP, in particular the knowledge of the transition model. However, this information is often unavailable. In this exercise, we consider the case in which a simulator is available but the transition probabilities are unknown.

Consider the simple grid world reported in Fig. 2. The state space is $\mathcal{S} = \{s_0, \dots, s_{10}\}$ and the action space is given by the four cardinal actions $\mathcal{A} = \{\text{right}, \text{down}, \text{left}, \text{up}\}$. The actions are stochastic, i.e., they may move the agent to an undesired state. In each state, the agent can execute only a subset of actions (i.e., actions leading against the wall are removed). For example,

$$\mathcal{A}(s_{10}) = \{\text{left}, \text{up}\} \quad \text{and} \quad \mathcal{A}(s_4) = \{\text{down}, \text{up}\}.$$

The cell filled with lines represents a wall, while target states are denoted in blue. The reward function is zero everywhere except in the target states ($r(s, a, s_3) = 1$ and $r(s, a, s_6) = -1$, for any (s, a)). The target state is absorbing and a single action is defined. Consider a discount factor of $\gamma = 0.95$.

A simulator of the MDP is available online (`gridworld.py` or `GridWorld.m`).

2.1 A Review of RL Agent/Environment Interaction

In this section, we review the process of interaction between agent and environment. We consider without loss of generality episodic settings¹ and that a simulator of the MDP is available.

The MDP simulator is in general composed by

1. **reset**: generates the initial states accordingly to the initial state distribution μ_0 .
2. **step**: receives in input a state and an action and it returns the next state, reward and a terminal flag. The terminal flag is true if and only if the state that is reached is absorbing. For example, in the grid world it is true when x_3 and x_6 are reached.

¹Non-episodic problems can be viewed as episodic problems with one episode.


s_0	s_1	s_2	s_3
s_4		s_5	s_6
s_7	s_8	s_9	s_{10}

Figure 2: [Exercise 2] Simple grid world with zero-based state enumeration. Since Matlab used one-based indices, in the provided code, states are numbered from 1 to 11.

Note that when an absorbing state is reached, the environment restarts from a state drawn accordingly to the initial state distribution. The overall structure of the interaction is reported in Alg. 1.

This schema is the building block of almost all the RL approaches. It can be used to collect trajectories ($\tau = \{(s_i, a_i, r_i, s_{i+1})\}_{i \geq 0}$) or to compute any estimate required by the RL algorithm (e.g., temporal difference error, MC estimate of value function).

Note on T_{\max} . In several episodic problems, it is easy to select the value for T_{\max} but, in general, it is not clear when to stop the simulation. In infinite-horizon discounted problems, we can use the notion of effective horizon: $\mathcal{O}(1/(1 - \gamma))$. In particular, if the reward is bounded $r_t \in [0, R_{\max}]$, by setting $T_{\max} = \mathcal{O}(-\log(\delta/R_{\max})/(1 - \gamma))$, the discounted truncated sum of rewards is δ -close to the infinite sum [Kakade et al., 2003, Sec. 2.3.3].

2.2 Work to do

1. **Policy evaluation.** Consider the deterministic policy that is selecting the action *right* when available, otherwise *up*.

Q4: denote with $V_n(x, a)$ the value function estimated using Monte-Carlo, i.e., empirical average:

$$V_n(s) = \frac{1}{N(s)} \sum_{k=1}^{N(s)} \left[\sum_{t=1}^{T_{\max}} \gamma^{t-1} r_t^{(k)} \right], \text{ with } s_1 = s, a_t \sim \pi(\cdot | s_t) \quad (1)$$

where $\sum_s N(s) = n$ and $(r_t^{(k)})$ is the sequence of rewards obtained when simulating the k -th trajectory (using the simulator).

Build such estimator and plot $J_n - J^\pi$ as a function of n , where

$$V^\pi = [0.877, 0.928, 0.988, 0, 0.671, -0.994, 0, -0.828, -0.877, -0.934, -0.994]^T$$

Algorithm 1: Agent/Environment interaction

```

1 for  $e = 1, \dots, E$  do
2    $s_0 = \text{reset}()$ 
3    $t = 0$ 
4   repeat
5      $a_t \sim \pi(\cdot | s_t)$ 
6      $s_{t+1}, r_t, \text{term} = \text{step}(s_t, a_t)$ 
7   until  $(t < T_{\max} \wedge \text{not term})$ 
8 end
```

is the value function computed with DP and

$$J_n = \sum_{s \in \mathcal{S}} \mu_0(s) V_n(s), \quad \text{and} \quad J^\pi = \sum_{s \in \mathcal{S}} \mu_0(s) V^\pi(s).$$

Note: μ_0 is the starting state distribution, you can estimate it using the reset function. Use an arbitrary number of samples to estimate it. Do this before to implement the MC approach.

Note: Functions for the visualization of the Q-function and policy are provided.

2. Policy optimization: *the Q-learning algorithm.*

Q-Learning estimates the quantities $Q^*(x, a)$ for all $(s, a) \in \mathcal{X} \times \mathcal{A}$. The algorithm simulates trajectories $\{\tau_i\}$ and updates its estimates along the way using

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

where

- $N(x, a)$ is the number of visits of the state-action (x, a)
- for each (x, a) , $\alpha_i(x, a)$ is a sequence of *stepsizes*
- in state x_t , a_t is chosen based on some *exploration policy*

For the algorithm to converge, the stepsizes and exploration policy should be chosen such that all state-action pairs are visited infinitely often and $\alpha_i(x, a)$ satisfies the usual stochastic approximation requirements (Robbins-Monro conditions [Robbins and Monroe, 1951]):

$$\sum_i \alpha_i(x, a) = +\infty \quad \text{and} \quad \sum_i \alpha_i^2(x, a) < +\infty.$$

You will implement Q-Learning in *episodes*² of length T_{\max} , with an ϵ -greedy exploration policy within each episode, i.e., select an action as $a_t = \arg \max Q(x_t, a)$ with probability $1 - \epsilon$ and randomize with probability ϵ .

Q5: Describe the (parameters of the) exploration policy and learning rate chosen, and illustrate the convergence of Q-Learning using the following performance metrics:

- Performance over all the other state $\|v^* - v^{\pi_n}\|_\infty$, where π_n is the greedy policy w.r.t. Q_n at the end of the n -th episode
- Reward cumulated over the episode

Note that

$$v^* = [0.877, 0.928, 0.988, 0, 0.824, 0.928, 0, 0.778, 0.824, 0.877, 0.828]^T.$$

3. **Q6:** Is the **optimal policy** of an MDP affected by the change of the initial distribution μ_0 ?

References

- Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.
- Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471619779.
- Herbert Robbins and Sutton Monroe. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

²*Note:* pay attention to how terminal states are treated.