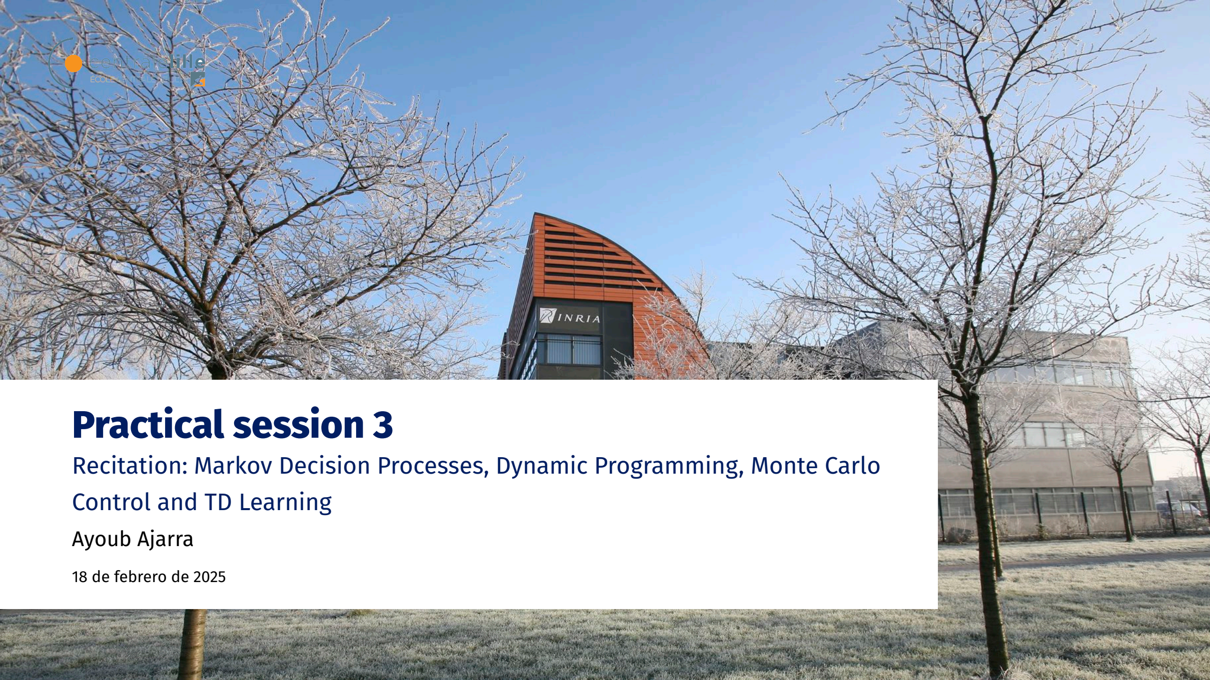


Practical session 3

Recitation: Markov Decision Processes, Dynamic Programming, Monte Carlo Control and TD Learning

Ayoub Ajarra

18 de febrero de 2025

Reminder: Markov Decision Process

Definition

An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where:

- $\mathcal{S}$ a set of states of the world.
- $\mathcal{A}$ a actions
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ state-transition function (Gives $p(s_{t+1}|s_t, a_t)$).
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ reward function (Gives $\mathbb{E}_{\mathcal{R}}\left\{\mathcal{R}(s_t, a_t)|s_t, a_t\right\}$).

Markov property

$p(r_t, s_{t+1}|s_0, a_0, r_1, \cdots, s_t, a_t) = p(r_t, s_{t+1}|s_t, a_t)$, $\langle \text{next state, expected reward} \rangle$ depends through the whole history only on $\langle \text{previous state, current action} \rangle$

Given dynamics, how to find an optimal policy?

Goal: solve an MDP by finding an optimal policy

- What is the objective?
  1. Reward: discounting and design.
  2. Expected objective: state and action-value function
- How to evaluate the objective?
  1. Bellman expectation equations.
- How to improve the objective?
  1. Bellman optimality equations
- Combine evaluation and improvement:
  1. Generalized Policy Iteration

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal.

Cumulative reward also referred to as return is:

$$G_t = R_t + R_{t+1} + \cdots + R_T$$

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal.

Cumulative reward also referred to as return is:

$$G_t = R_t + R_{t+1} + \cdots + R_T$$

Natural rewards exist for many application:

- Maze solving: -1 every time step until the agent escapes.
- Chess: +1 for winning, -1 for losing, 0 for drawing the games.

Critical that the rewards we set up indicate what we want the agent to accomplish and not how to achieve it.

Explaining goals to agent through reward

centralelille
ÉCOLE CENTRALE DE LILLE

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal.

Cumulative reward also referred to as return is:

$$G_t = R_t + R_{t+1} + \cdots + R_T$$

Natural rewards exist for many application:

- Maze solving: -1 every time step until the agent escapes.
- Chess: +1 for winning, -1 for losing, 0 for drawing the games.

Critical that the rewards we set up indicate what we want the agent to accomplish and not how to achieve it. ▸ Reward hacking examples.

Example 1: Continuous Cooling System for Data Centers

- States – temperature measurements
- Actions – different fans speed.
- $R = 0$ for exceeding temperature thresholds
- $R = +1$ for each second system is cool.

Example 1: Continuous Cooling System for Data Centers

- States – temperature measurements
- Actions – different fans speed.
- $R = 0$ for exceeding temperature thresholds
- $R = +1$ for each second system is cool.

What could go wrong with such a design?

Example 1: Continuous Cooling System for Data Centers

- States – temperature measurements
- Actions – different fans speed.
- $R = 0$ for exceeding temperature thresholds
- $R = +1$ for each second system is cool.

What could go wrong with such a design?

Infinite return for non-optimal behavior!

Example 2: Robot motion (reaching destination Z)

- State – position, velocities of joints
- Actions – actuator forces to joints
- Reward $R = max(0, d(x, Z) - d(x', Z))$

Example 2: Robot motion (reaching destination Z)

- State – position, velocities of joints
- Actions – actuator forces to joints
- Reward $R = max(0, d(x, Z) - d(x', Z))$

What could go wrong with such a design?

Example 2: Robot motion (reaching destination Z)

- State – position, velocities of joints
- Actions – actuator forces to joints
- Reward $R = max(0, d(x, Z) - d(x', Z))$

What could go wrong with such a design?

Positive feedback loop!

Reward discounting

Reward discounting

Idea: Get rid of infinite sum by discounting

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \cdots + R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

Intuition:

Reward discounting

Idea: Get rid of infinite sum by discounting

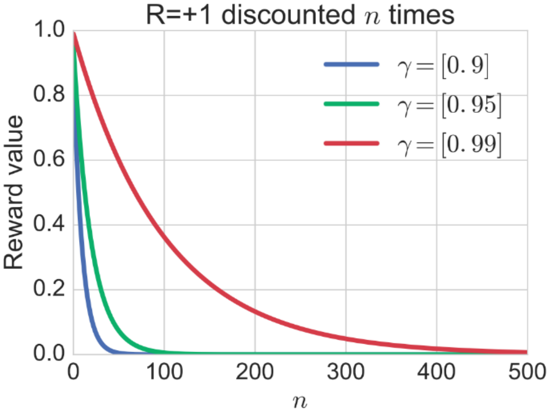
$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \cdots + R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

Intuition: The same cake compared to today's one worth:

1. γ times less tomorrow
2. γ^2 times less the day after tomorrow
3. $\cdots etc$

Discounting makes return finite

Maximal return for $R = +1$: $G_0 = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$



Discounting is inherent to humans [1]

- Quasi-hyperbolic $f(t) = \beta\gamma^t$
- Hyperbolic discounting $f(t) = \frac{1}{1+\beta t}$
- Some ideas in economics: value of $100 is higher today than in the future.
- Future is uncertain: reduce its influence for making decisions at the current time step.

[1] Laibson, D. (1997). Golden eggs and hyperbolic discounting. The Quarterly Journal of Economics, 112(2), 443-478.

Finding optimal policy

Solving the MDP

Solving the MDP means finding the sequence of actions with the largest (discounted) return.

Definition

A policy is a mapping from a trajectory to an action.

$$\pi : \mathcal{H} \to \Delta(\mathcal{A})$$

Remarks:

- π is said to be stationary if it depends only on the current state (i.e. $\pi : \mathcal{S} \to \Delta(\mathcal{A})$)
- π is said to be deterministic if the output is an action (i.e. $\pi : \mathcal{S} \to \mathcal{A}$)

State value function

Definition

The value of a state s under a policy π denoted by $V^{\pi}(s)$ is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{G_t | S_t = s\}$$

Remark:

The value function at the terminal state is zero (end of interaction).

Action value function

Definition

The value of taking action a in a state s under a policy π denoted by $Q^{\pi}(s)$ is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\{G_t | S_t = s, A_t = a\}$$

The expected total reward agent gets from state s by taking action a and following policy π from the next state.

Action value function

Definition

The value of taking action a in a state s under a policy π denoted by $Q^{\pi}(s)$ is defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\{G_t | S_t = s, A_t = a\}$$

Assuming I know the state value, how to compute the action value? and vice versa

$$Q^{\pi}(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma V^{\pi}(s'))$$

$$V^{\pi}(s) = \sum_{a} \pi(a|s) Q^{\pi}(s,a)$$

Bellman optimality equations

Bellman equation for value function

$$V^*(s) = \max_a \mathbb{E}\{R_t + \gamma V^*(s_{t+1}) | S_t = s, A_t = a\}$$

Alternatively,

$$V^*(s) = \max_a \sum_{s',r} p(s', r | s, a)(r + \gamma V^*(s'))$$

Bellman equation for action-state value function

$$Q^*(s,a) = \max_a \mathbb{E}\{R_t + \gamma \max_{a'} Q^*(s_{t+1}, a') | S_t = s, A_t = a\}$$

Alternatively,

$$Q^*(s,a) = \sum_{s',r} p(s', r | s, a)(r + \gamma \max_a Q^*(s', a'))$$

Solving Bellman equation: Dynamic programming

The idea is to turn Bellman optimality equations into update rules in two steps:

1. Policy evaluation\prediction: Given a policy π, how to estimate $V^{\pi}(s)$?
2. Policy improvement: Given the estimated $V^{\pi}(s)$, how a policy such π' s.t $\pi' \geq \pi$ (order defined by the value function).

This is referred to as policy iteration.

Step 1: Policy evaluation

Given a policy π, compute V^π.

Update rule

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\{G_t | S_t = s\} \\
&= \mathbb{E}_\pi\{R_{t+1} + \gamma G_{t+1} | S_t = s\} \\
V^\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)\{r + \gamma V^\pi(s')\} \\
V_{n+1}(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a)\{r + \gamma V_n(s')\}
\end{aligned}
$$

Algorithm

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Policy improvement

Policy improvement theorem

Let (π, π') denote a pair of deterministic policies s.t:

$$\forall s \in \mathcal{S} : Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s)$$

Then $\pi' \geq \pi$.

Combining the two steps: policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   - $\Delta \leftarrow 0$
   - Loop for each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Hands on code

What we've learned so far ...

We assumed that dynamics are known (**Model-based RL**). In this case Dynamic Programming can be applied and we can plan ahead.
In real-world scenarios, this is not always true.

Dealing with unknown dynamics

Assuming I know the What can we do when the dynamics ($p(s'|s,a)$) are unknown?

What we've learned so far ...

We assumed that dynamics are known (**Model-based RL**). In this case Dynamic Programming can be applied and we can plan ahead.
In real-world scenarios, this is not always true.

Dealing with unknown dynamics

Assuming I know the What can we do when the dynamics ($p(s'|s, a)$) are unknown?

Model-free RL

We can sample trajectories and try random actions ...

Approach 1: Monte-Carlo

Idea

1. Sample all trajectories conditions on current state action (s, a).
2. Loop over generated trajectories to estimate the returns conditioned on the current state action.
3. Take the average over the trajectories collected.

Approach 1: Monte-Carlo

Idea

1. Sample all trajectories conditions on current state action (s, a).
2. Loop over generated trajectories to estimate the returns conditioned on the current state action.
3. Take the average over the trajectories collected.

Many trajectories to deal with ...

This algorithm requires lot of computations, and does not work in more complex environments (For example in high dimensional state space, images,...)

Alternatives: TD-learning (On board)

1. Q-learning.
2. SARSA
3. Expected SARSA
4. A comparison between Q-learning and SARSA (Q-learning achieves optimality without further exploration.)

Questions ?