Introduction to Reinforcement Learning
3/10

Jean Martinet

MSc DSAI

2024 – 2025

Programme

- Introduction
  - Course 1 : Introduction to Reinforcement Learning (RL)
- Part I on tabular methods
  - Course 2 : Markov Decision Processes
  - **Course 3 : Dynamic programming in RL**
  - Course 4 : Temporal difference 1/2 (Q-learning)
  - Course 5 : Temporal difference 2/2 (SARSA)
- Part II on approximate methods
  - Course 6 : Value function approximation
  - Course 7 : Eligibility traces
  - Course 8 : Policy gradient 1/2 (REINFORCE)
  - Course 9 : Policy gradient 2/2 (actor-critic methods)
  - Course 10 : Projects presentation session

Reminder : think of a project topic

- **Choose from :**
  - Public presentation of articles/advanced topics/applications
    - Conference paper or book chapter
    - Advanced theme (e.g. actor-critic, eligibility trace, etc.)
    - Application domain (e.g. temperature control, revenue management, etc.)
  - Deepening or exploration project
    - Subject to be chosen/defined and validated
- **Choice to be validated before session 4**
- **Expected result :**
  - Short 2-page max PDF report
  - Code (ipynb / py / git)
  - Short 10-min presentation during last / before last session

About the project

- Double objective
  - Dig deeper in a specific subject (discussed or not during the lectures)
  - Share your insights with other students (in a teacher mode)
- A bit hard to choose early, before having reviewed all topics
- If you can define what is the environment, the reward, the agent, and the actions, it is a good start
- Stay small, at least for a first version, then make it more complex if you have time
- An experimental contribution is needed
  - E.g. compare two algorithms
  - E.g. start from an existing approach, and monitor changes when parameters vary
- The project needs be ORIGINAL
  - You need an original contribution of your own
  - Make sure your project is different from what can be found online
- IMPORTANT : if you decide to use an existing work, it is MANDATORY to cite the source, and you need to state what your contribution is

Note

- Graded lab next week (Q-learning)

Today's menu

- Key idea of DP
- Policy iteration (evaluation-improvement)

Dynamic programming

- Collection of algorithms that can be used to compute optimal policies given a perfect model of the environment like MDP
- Limited utility – yet theoretically important
  - Perfect model assumption
  - Computationally expensive
- DP uses value functions to organise the search for good policies

Remember last week

- Bellman optimality equation for V^*

$$V^*(s) = \max_a \mathbb{E}[r_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma V^*(s')]$$

- Bellman optimality equation for Q^*

$$Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r | s, a)[r + \gamma \max a' Q^*(s', a')]$$

- We can easily find π^* once we have once we have V^* or Q^*
- DP algorithms turn Bellman equations into update rules

Policy evaluation (prediction)

- How to compute V^π for an arbitrary π ?
- Remember last week

$$\begin{aligned} V^\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi(s')] \end{aligned}$$

- If the environment dynamics are completely known, we can iterate :

$$\begin{aligned} V^\pi_{k+1}(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi_k(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^\pi_k(s')] \end{aligned}$$

- Initialisation : random for all states, expect 0 terminal states
- We are guaranteed to converge towards V^π when $k \to \infty$
- This is the *iterative policy evaluation*

Policy evaluation (prediction)

- Each iteration updates all cells
  - Two versions : 2 arrays versus in-place (equivalent, the latter is faster)

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

- The objective of estimating V^π is to improve π
  - From s, we know how good it is to follow π : it is $V^\pi(s)$
  - What if we took another action $a \neq \pi(s)$?
- One way to answer : select a and then follow π

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r | s, a)[r + \gamma V^\pi(s')] \end{aligned}$$

- Key criterion : is it greater than or less than $V^\pi(s)$?
  - If greater, change π so as to always select a in s
  - We obtain a new greedy policy π'

$$\begin{aligned} \pi'(s) &\doteq \arg\max_a Q^\pi(s,a) \\ &= \arg\max_a \mathbb{E}[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \arg\max_a \sum_{s',r} p(s', r | s, a)[r + \gamma V^\pi(s')] \end{aligned}$$

Policy improvement theorem

- Let π and π' be any pair of deterministic policies such that

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \text{ for all } s$$

- Then π' must be as good or better than π

$$V^{\pi'}(s) \geq V^\pi(s) \text{ for all } s$$

- Note that if $V^\pi = V^{\pi'}$
  - $V^{\pi'}$ must be V^*, and
  - π and π' must be optimal policies

Policy iteration

- Once a policy π has been improved using V^{π} to obtain the better π', we can compute $V^{\pi'}$ and improve it again to obtain π''
- Sequence of monotonically improving policies

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^{\pi^*}$$

- Finite MDP has finite number of policies, so finite number of iterations
- This is called *policy iteration*

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 $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s',r \mid s,\pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r \mid s,a) \big[r + \gamma V(s')\big]$
   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

Today's lab

- Find $\pi*$ for TicTacToe using DP
  - Check https://medium.com/@nour.oulad.moussa/tic-tac-toe-with-reinforcement-learning-part-i-markov-deci for help
- (Note : Gymnasium next next week)