Introduction to Reinforcement Learning
4/10

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... today**
- **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

Today's menu

- Monte Carlo methods
- Dynamic programming vs Temporal difference
- On-policy vs off-policy learning
- Exploration vs exploitation
- Optimal policy approximation with Q-learning

Monte Carlo methods

- MC does not assume a complete knowledge of the environment
  - MC methods require only *experience*, contrary to DP
  - A model is required
  - Obtain optimal behaviour without any prior information about environment dynamics
    - It is sometimes easier to generate transitions than to obtain complete probability distributions in explicit forms
- The method is based on averaging sample returns
  - Strong assumption : episodic tasks only – to insure that well defined returns are available
  - Only after episode completion, value estimates and policies are changed – not step-by-step updates

MC prediction

- Prediction means : "evaluate policies"
- Simple average of returns observed after visits to each state
- Converges to the true expected values when the number of visits to s goes to infinity
- Two-versions : *first-visit* and *every-visit*

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

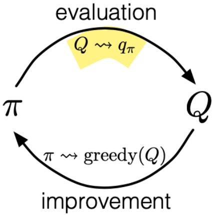
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

Evaluation of action values

- With a model, state values alone are sufficient to determine a policy
  - Just select the action that leads to the best (s_{t+1}, r_{t+1}) as in DP
- When a model is not available, we need to explicitly evaluate the value of actions
  - Estimate $Q^\pi(s, a)$ similarly, except that we now deal with (s, a) visits
- However, many (s, a) pairs may never be visited if π is deterministic
  - Option to *force* the episode to start with all possible (s, a)
  - This is the *exploring starts* assumption – in simulation not real world

MC control

- Control means : "approximate optimal policies"
- Generalised Policy iteration
  - Greedy policy with the same pattern as DP with $\pi(s) \doteq \arg\max_a Q^\pi(s, a)$



$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*$$

Temporal difference

- Simplest TD update

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

- The amount in brackets is an error measuring the difference between
  - the estimated value of S_t and
  - the better estimate $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
- This quantity is called the *TD error*

$$\delta_t \doteq r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Tabular TD(0)

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\textit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Dynamic programming vs Time difference

- In DP, we assumed a complete knowledge of the environment (MDP)
  - We *computed* value functions
  - Now with TD, we *learn* them
- We require only *experience*
  - Actual : no prior knowledge of environment is required
  - Simulated : a model is required to generate sample transitions
  - (it is easier to get sample transitions than the complete distribution)
- Everything else remains similar

On-policy vs off-policy

- Our agent learns action values relying on our current policy estimate
- But it needs to behave non-optimally to explore all actions
- (and find optimal actions)
- We can use – and distinguish – two policies
  - The policy that we learn and becomes optimal (target π)
  - The exploratory policy that generates behaviour (behaviour π)
- This is *off-policy* learning
- On the contrary, *on-policy* methods use a single policy

Greedy actions vs exploratory actions

- Simplest action selection rule : select one of the actions with the highest estimated value

$$A_t \doteq \arg\max_a Q^\pi(s, a)$$

- Greedy action selection exploit current knowledge to maximise immediate reward
- Alternative :
  - Behave greedily most of the time
  - Sometimes, with probability ϵ, randomly pick another action (exploratory move)
- This is the ϵ-greedy method

ϵ-greedy action selection

- Trade-off between exploration and exploitation using ϵ value
- ϵ starts at 1 (only exploration) then decreases (more exploitation)
- Choose a random number r between 0 and 1 (uniform distribution) :
    - if $r < \epsilon$: choose a random action (exploration)
    - if $r \geq \epsilon$: choose the best action = maximising Q value (exploitation)

Q-learning algorithm

- Here again, π evaluation (prediction) and π improvement towards π^* (control)
- Q-learning is an off-policy TD algorithm

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Iteratively updates action-value using δ_t
- α is the learning rate
- Build a Q-table of size $|S| \times |A|$, initial values are 0s
- Iterate episodes (a few hundreds)
  - Iterate steps in each episode
    - Select the best action a in state s, use the reward to update Q
    - Episode terminates when agent reaches a terminal state (or max iteration)

Today's lab (graded, duration 1h30)

- 1. Consider a 1D grid with :
  - one goal location (positive reward, e.g. $+1$)
  - one trap location (negative reward, e.g. -1)
  - a fixed move cost (e.g. -0.01)
  - deterministic actions (probability to go left when trying left is 1)
- Implement Q-learning from the equation, with ϵ-greedy
- Run your algorithm to determine the best policy, check it
- 2. Extend to the 2D grid of the classical toy example (lecture 1)
- 3. Optional : Visualise results with plots for several values of γ, α, and ϵ (brings bonus !)
- (Note : use Python and numpy only – gym next week)
- Submit your solution after 90 min