Faculté
des **sciences économiques** et de **gestion**
Université de Strasbourg

Reinforcement Learning

UE 2 Machine learning

Nattirat Mayer

Fall-Winter Semester 2025-26

Chapter 5. Monte Carlo Methods

Here comes a new chapter!

5. Monte Carlo methods

- In Chapter 4 - DP, we need complete knowledge of the environment's dynamics, the transition probabilities $p(s', r|s, a)$
- This chapter, Monte Carlo (MC) Methods, use averages to approximate values
- MC learns directly from episodes of experience
- MC learns from complete episodes. Thus, MC can only apply episodic MDPs, i.e., all episodes must terminate

Model-Based RL

Dynamic Programming
-Policy Iteration
-Value Iteration

Model-Free RL

Monte Carlo
-On-Policy MC
-Off-Policy MC

Model-Free RL

Temporal Difference
- TD(.) (on-policy)
-SARSA (on-policy)
-Q-Learning (off-policy)

Model-Free RL

Deep RL
- Deep Q-Learning (off-policy)
- Policy Gradient (on-policy)

*Not exclusive list

Model-Based and Model-Free Methods in RL

- **Model**: Anything the agent uses to predict the environment's response to its actions. To predict the environment, the agent uses the transition probability $p(s, r|s, a)$.
- **Model-Based**: Methods which use a model to plan actions before they are taken.
  - E.g. in Checker, the agent uses predictions of the opponent's moves to determine its 'best' (optimal) move before making the move.
- **Model-Free**: Methods without a model. They learn action-to-return associations. That is, it is more useful to estimate $q(s, a)$ than $v(s)$.

  In Model-Free, why we use $q(s, a)$, and not just $v(s)$?

This is because $q(s, a)$ tells the agent what action to take given the state s. From the **Bellman optimality condition** for q_* in chapter 3 MDP. We have

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)\big[r + \gamma \max_{a'} q_*(s', a')\big].$$

But in Model-Free, we do not know $p(s', r|s, a)$. Therefore, we estimate $q(s, a)$ directly.

Sampling and MC

Reference: cognitiveclass.ai, 2025

- In RL, sampling means 'playing out full episodes'.
- MC uses sampling method which allows the agent to learn from actual experience
- Unlike DP (last Chapter), MC uses **sampling** to estimate value functions through sampling episodes of experience

Assumptions:

- All episodes must terminate before the return G_t can be computed
- Need to run sufficiently large number of episodes for accurate estimates (law of large numbers)

Many algorithms of MC

- On-policy MC
    - Monte Carlo ES (Exploring Starts)
    - On-policy first-visit MC Control
- Off-policy MC
    - Off-policy MC prediction
    - Off-policy MC control

5.1 Exploration and Exploitation Trade-Off in MC

MC methods learn $v(s)$ by playing out full episodes (sampling), thus it's important that every state-action pair are visited.

- To discover optimal policies π_*, we must **explore** all state-action pairs
- To get high returns G_t, we must **exploit** the high value pairs we know that give us high G_t

How to implement Exploration and Exploitation Trade-Off in MC?

With infinite data, optimal policy π_* is always discouverable if the policy is **SOFT** ($\epsilon-$soft policy). That is,

$$\pi(a \mid s) > 0 \quad \text{for all } s \in \mathcal{S},\ a \in \mathcal{A}(\int)$$

We implement $\epsilon -$ *Greedy* policy as one of the mechanism for Exploration and Exploitation Trade-off. Recall,

- ϵ is the degree of **randomness**
- In general, $\epsilon = 0.1$ means we explore 10% of the time or $\epsilon = 0.01$ means we explore 1% of the time

$\epsilon -$ *Greedy* policy of Q: With probability ϵ, take an action selected randomly (and uniformly) from action space $\mathcal{A}$, otherwise take $\arg\max_a Q(s, a)$

5.4 On-Policy vs. Off-Policy Learning

The agent must both **explore** and **learn**. Another approach to encourage explorations is on-policy and off-policy learning. We first introduce

Behavior policy: Generate the data. It is denoted as

$$b(a \mid s)$$

Target policy: To be improved/evaluated. It is denoted as

$$\pi(a \mid s)$$

On-Policy Learning $b = \pi$

- "*Learn on the job*"
- Learn about policy π from experience sampled from target policy π
- Learns from its own (ϵ-greedy) experience

Off-Policy Learning $b \neq \pi$

- "*Look over someone's shoulder*"
- Learn about policy π from experience sampled from behavior policy b
- Learns about an optimal greedy policy while behaving ϵ-greedy

Example of On- and Off-Policy

- The rover operates in a 1D world with **7 states**: $s_1, s_2, \ldots, s_7$.
- The goal is at s_7
- $\mathcal{A} = \{\mathit{left}, \mathit{right}\}$

s_1	s_2	s_3	s_4	s_5	s_6	s_7
						

Reference: Brunskill (2024) - CS234 Reinforcement Learning

On-Policy MC

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Off-Policy MC

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

5.5 Off-policy Prediction via Importance Sampling

What is W ?

We want to estimate $q_\pi(s, a)$, the action-value function for a **target policy** π, but our episodes come from a **behavior policy** $b \neq \pi$.

Not all episodes are equally informative. Actions taken by b may be more or less likely under π. We adjust each episode's contribution using a **weight** W:

$$W_t = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{b(a_k|s_k)}$$

where W_t is cumulative importance weight from step t to end of episode T.

Update Rule (weighted average):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{W_t}{C(s_t, a_t)} \Big[G_t - Q(s_t, a_t) \Big]$$

where G_t = return from step t to episode end, and $C(s_t, a_t)$ is cumulative sum of weights for (s_t, a_t).

In python,

```
G = 0.
W = 1.
# Loop inversely to update G and Q values
while trajectory:
    (state, action, reward, act_prob) = trajectory.pop()
    G = gamma * G + reward
    C[state][action] = C[state][action] + W
    Q[state][action] = Q[state][action] + (W / C[state][action]) * (G - Q[state][action])
```

On-Policy Learning

- On-policy methods are generally simpler and are considered first

Off-Policy Learning

- Off-policy methods are often of higher variance and are slower to converge
- However, off-policy methods are more powerful and general

5.7 MC Tutorial

Off-Policy MC Control Algorithm to Racetrack

This tutorial is from Ou, 2023 - Towards Data Science

The tacks ...

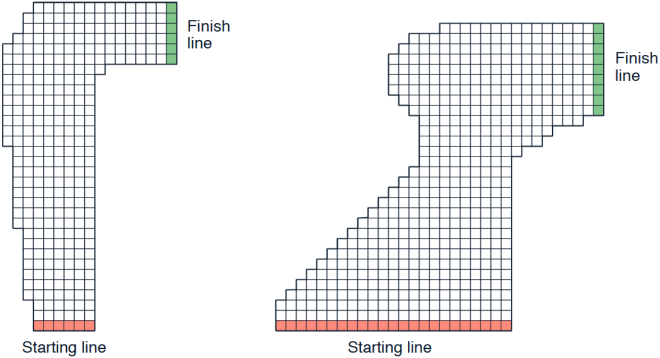

Figure 5.5: A couple of right turns for the racetrack task.

State space $\mathcal{S}$:

- Each state s: (s_x, s_y, v_x, v_y)
- (s_x, s_y) – position on track grid
- (v_x, v_y) – velocity components, $0 \leq v_x, v_y \leq 5$

Action space $\mathcal{A}$:

- Acceleration $(a_x, a_y) \in \{-1, 0, +1\}^2$
- Total of 9 actions (change in velocity)
- Velocity clipped to $[0, 5]$ each step

Stochastic transition: With prob. 0.1, acceleration ignored ("slip")

Reward:

- -1 per step until finish line
- Episode ends when finish or off-track

5.8 Downsides of Monte Carlo Methods

Monte Carlo agents learn slowly since

- MC must wait until end of episode before return is known
- MC can only learn from complete sequences
- MC only works for episodic (terminating) environments