Lecture 7: Monte Carlo for RL
A finite Markov Decision Process (MDP) is a tuple \((S, A, P, R, \gamma)\) where:

- \(S\) is a finite set of states
- \(A\) is a finite set of actions
- \(P\) is a state transition probability function
- \(R\) is a reward function
- \(\gamma\) is a discount factor
Methods Overview

- Dynamic Programming Methods:
  - *require a model*
  - *bootstrap*

- Monte Carlo Methods:
  - *do not* require a model
  - *do not* bootstrap

- Temporal-Difference Learning Methods:
  - *do not* require a model
  - *bootstrap*
Today’s Content

- Coursework 1 (questions/discussion)
- Monte Carlo (MC) Policy Evaluation
  - State-value and Action-value functions
- MC Control
- MC Exploring Starts
- Problems with MC Assumptions
- On-Policy MC Control
Coursework 1

- **Actions:**
  - UP_LEFT
  - UP
  - UP_RIGHT

- **Episodic task**
  - Top row (row 1) = terminal states

- **Transition Function:**
  - `MDP_1.getTransitions`

- **Reward Function:**
  - `MDP_1.getReward`

- **Policy to evaluate:**
  - `pi_test1 / pi_test1_stateNumbers`

[Row, column] coordinates numbered left→right and top→bottom
Monte Carlo Methods

- **Learn** value functions
- **Discover** optimal policies
- Do not assume knowledge of model as in DP, i.e., $P_{ss'}^a$, $R_{ss'}^a$

- Learn from experience: Sample sequences of states, actions and rewards ($s, a, r$)
  - In simulated or real (e.g., physical robotic) worlds
  - Clearly, simulator *is* a model but not a *full* one as in a prob. distribution

- Eventually attain optimal behaviour (same as with DP)
Backup in MC

- Does the concept of backup diagram make sense for MC methods?
- As in figure, MC does not sample all transitions
  - Root node to be updated as before
  - Transitions are dictated by policy
  - Acquire samples along a sample path
  - Clear path from eventual reward to states along the way (credit assignment easier)
- Estimates are different states are independent
  - Computational complexity **not** a function of state dimensionality
Pictorial: What does DP Do?

\[ V(s_t) \leftarrow E_{\pi} \{ r_{t+1} + \gamma V(s_{t}) \} \]
Pictorial: What does Simple MC Do?

\[ V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \]

where \( R_t \) is the actual return following state \( s_t \).
Monte Carlo Policy Evaluation

- **Goal**: Approximate a value function $V^\pi(s)$
- **Given**: Some number of episodes under $\pi$ which contain $s$
- Maintain average returns after visits to $s$

- **First visit vs. Every visit MC**:
  - Consider a reward process $R(t) = r_t + \gamma r_{t+1} + \ldots$ and define the first visit time, $\tau = \min\{t|x = x_i\}$ and a set, $\Gamma = \{t|x = x_i\}$
  - First visit MC averages $\{R^i(\tau)\}, i = 1, \ldots, n$
  - whereas every visit MC averages over $\{R^i(t_j)\}, i = 1, \ldots, n, t_j \in \Gamma$

What is the effect of $\pi$?
What if it is deterministic?
First-visit Monte Carlo Policy Evaluation

Initialize:

\[ \pi \leftarrow \text{policy to be evaluated} \]
\[ V \leftarrow \text{an arbitrary state-value function} \]
\[ \text{Returns}(s) \leftarrow \text{an empty list, for all } s \in S \]

Repeat forever:

(a) Generate an episode using \( \pi \)
(b) For each state \( s \) appearing in the episode:
    \[ R \leftarrow \text{return following the first occurrence of } s \]
    Append \( R \) to \( \text{Returns}(s) \)
    \[ V(s) \leftarrow \text{average}(\text{Returns}(s)) \]
Example: Road Fighter

- So, at every state, we know what actions are available…
- but we don’t know anything of where we might transition, and with what probability…
- or what reward signals we might receive.
- Given a policy, we compute the average return starting from a state, across episodes.
- Obviously, the episodes need to terminate.
- Difference between first-time and any-time visit MC here?
Monte Carlo Estimation of Action Values

• Model is not available, so we do not know how states and actions interact
  – We want $Q^*$

• We can try to approximate $Q^\pi(s,a)$ using Monte Carlo method
  – Asymptotic convergence if every state-action pair is visited

• **Explore many different starting state-action pairs:** Equal chance of starting from any given state
  – Not entirely practical, but simple to understand
Monte Carlo Control

• **Policy Evaluation:**
  Monte Carlo method

• **Policy Improvement:**
  Greedify with respect to state-value of action-value function
Convergence of MC Control

• Policy improvement still works if evaluation is done with MC:

\[
Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, \arg \max_a Q^{\pi_k}(s, a)) \\
= \max_a Q^{\pi_k}(s, a) \\
\geq Q^{\pi_k}(s, \pi_k(s)) \\
= V^{\pi_k}(s).
\]

• \(\pi_{k+1} \geq \pi_k\) by the policy improvement theorem

• Assumption: exploring starts and infinite number of episodes for MC policy evaluation (i.e., value function has stabilized)

• Things to do (as in DP):
  – update only to given tolerance
  – interleave evaluation/improvement
Monte Carlo Exploring Starts

Initialize, for all $s \in S$, $a \in A(s)$:
- $Q(s, a) \leftarrow$ arbitrary
- $\pi(s) \leftarrow$ arbitrary
- $\text{Returns}(s, a) \leftarrow$ empty list

Repeat forever:
(a) Generate an episode using exploring starts and $\pi$
(b) For each pair $s, a$ appearing in the episode:
   - $R \leftarrow$ return following the first occurrence of $s, a$
   - Append $R$ to $\text{Returns}(s, a)$
   - $Q(s, a) \leftarrow \text{average}(\text{Returns}(s, a))$
(c) For each $s$ in the episode:
   - $\pi(s) \leftarrow \arg\max_a Q(s, a)$

Fixed point is optimal policy $\pi^*$
Can We Avoid Thorny Assumptions?

• Two major MC assumptions (infinite sampling and exploring all states) are unrealistic. How to circumvent the issue?

• Need to continually explore, $\varepsilon$-soft policies:

  – **On-policy** method: Explore in an $\varepsilon$-greedy manner

  – **Off-policy** method: Use a behaviour policy that is good at exploring, then infer optimal policy from that
On-Policy Monte Carlo Control

• Overall idea is still that of Generalized Policy Iteration (move \textit{towards} greedy policy), but throw in continual exploration

• In order to always explore, we want to keep policy \textit{\ensuremath{\boldsymbol{\varepsilon}}-soft}:

\[
\pi(s, a) > 0, \forall s, \forall a
\]

• Moreover, one may really wish to adopt an \textit{\ensuremath{\boldsymbol{\varepsilon}}-greedy} policy:

\[
\pi(s, a) = \frac{\varepsilon}{|A|}, \text{ if } a \text{ is not the greedy choice}
\]

\[
= 1 - \varepsilon + \frac{\varepsilon}{|A|}, \text{ if } a \text{ is the greedy choice}
\]

• In this case, we have \( \pi(s, a) > \frac{\varepsilon}{|A|}, \forall s, \forall a \)
The Policy Improvement Step

- Any $\epsilon$-greedy policy w.r.t. $Q^\pi$ is an improvement over any $\epsilon$-soft policy $\pi$ (Policy Improvement Theorem)

$$Q^\pi(s, \pi'(s, a)) = \sum_a \pi'(s, a)Q^\pi(s, a)$$

$$= \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \epsilon) \max_a Q^\pi(s, a)$$

$$\geq \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + (1 - \epsilon) \sum_a \frac{\pi(s, a) - \frac{\epsilon}{|A(s)|}}{1 - \epsilon} Q^\pi(s, a)$$

This is bounded above by,

$$= \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) - \frac{\epsilon}{|A(s)|} \sum_a Q^\pi(s, a) + \sum_a \pi(s, a)Q^\pi(s, a)$$

$$= V^\pi(s)$$
On-Policy MC Control

Initialize, for all $s \in S$, $a \in A(s)$:
- $Q(s, a) \leftarrow$ arbitrary
- $Returns(s, a) \leftarrow$ empty list
- $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
(a) Generate an episode using $\pi$
(b) For each pair $s, a$ appearing in the episode:
   - $R \leftarrow$ return following the first occurrence of $s, a$
   - Append $R$ to $Returns(s, a)$
   - $Q(s, a) \leftarrow$ average($Returns(s, a)$)
(c) For each $s$ in the episode:
   - $a^* \leftarrow \arg \max_a Q(s, a)$

For all $a \in A(s)$:
- $\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|A(s)| & \text{if } a = a^* \\ \varepsilon/|A(s)| & \text{if } a \neq a^* \end{cases}$

Evaluates as before

Improve towards $\varepsilon$-greedy, not the max
Reading +

- Sections 5.1 to 5.4 of Sutton and Barto (1st Edition)

**Optional (will take you away from course material):**

- Section 3.2 of Ng, A. et al. (2004)
  Autonomous inverted helicopter flight via reinforcement learning