PAGE EDGE
Date : / /

Ch-3, Finite Markov Decision Processes

→ **Markov Decision Process :**

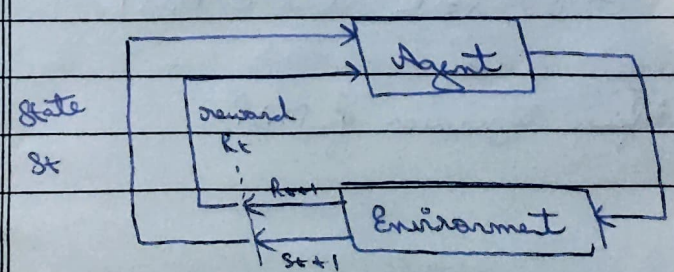
- Classical formalization of sequential decision making, where actions not only influence immediate rewards but also subsequent situations or state and ∴ future rewards.
- ∴ MDP involves delayed reward and need to tradeoff b/w immediate and delayed rewards.
- we estimate value $q_*(s,a)$ of each action a in each state s, or we estimate the value $v_*(s)$ of each give optimal action selections
- Idealized form of RL pose for which precise theoretical statements can be made.

The Agent - Environment Interface :-

→ Agent - learner and decision maker

Environment - The thing agent interacts with, comprising everything outside the agent.

→ They interact continuously, the agent selecting actions and the env responding with new situations and rewards.



Agent-Env interacti
is a Markov Decision Proces

→ The interaction b/w the agent and the env occurs at discrete time steps ($t = 0, 1, 2, 3 \dots$)

- At each time step t, the agent receives some representation of the env's state, $S_t \in S$ and on that basis selects one an action, $A_t \in A(s)^3$.

- One time step later, in part as a consequence of its action, the agent receives a numerical reward $R_{t+1} \in R \subset \mathbb{R}$ and finds itself in a new state, S_{t+1}

- This interaction generates a sequence of states, action and rewards:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \dots$$

→ In finite MDP, sets of actions, states and rewards (S, A, R) have a finite number of elements.

R_t, S_t – random variables with well defined discrete prob distribution dependent only on the preceding action and state

$$p(s', r | s, a) \doteq P_r \{ S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a \}$$

For particular values of RV, $s' \in S$ and $r \in R$, there is a prob of those values occuring at time t, given particular values of the preceding state and action.

for all $s, s' \in S$, $r \in R$ and $a \in A(s)$

- finite p – defines dynamics of MDP

PAGE EDGE
Date : / /

- The dot over the equal sign in the eq^n reminds us that it is a definition rather than a fact that follows from previous definitions

- Dynamic function $p: S \times R \times S \times A \to [0,1]$ is a ordinary deterministic function of 4 arguments

→ $\sum_{s' \in S} \sum_{r \in R} p(s', r | s, a) = 1$ for all $s \in S, a \in A(s)$

p – probability distribution for each choice of s and a

→ In Markov decision process, the prob given by p completely characterizes the env's dynamics

That is, prob of each possible value for S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and given them, not at all on earlier states and actions

This can be best viewed a restriction not on the decision process, but on the state

* State must include info about all aspects of the past agent - env interaction that makes a difference for the future = Markov property

→ From the 4 arg dynamics function p, one can compute anything else one might want to know about the env, such as the state-transition prob (which we denote, with a slight abuse of notation, as 3 arg function $p: S \times S \times A \to [0,1]$)

$$p(s' \mid s,a) = P_r(S_t = s' \mid S_{t-1} = s,\ A_{t-1} = a)$$

$$= \sum_{r \in R} p(s', r \mid s, a)$$

- Compute reward for state action pair as a 2 argument function $r : S \times A \to R$:

$$r(s,A) \doteq E[R_t \mid S_{t-1} = s,\ A_{t-1} = a]$$

$$= \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a)$$

The expected reward for state-action-next state triple as a 3 arg function $r : S \times A \times S \to R$

$$r(s,a,s') = E[R_t \mid S_{t-1} = s, A_{t-1} = a,\ S_t = s']$$

$$= \sum_{r \in R} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

→ MDP framework - abstract & flexible and can be applied to many different problems in many different ways.

- The steps, actions, states can be defined in various ways to suit different tasks.

Note: Actions can be any decisions we want to learn how to make

State can be anything we know that might be useful in making them

PAGE EDGE
Date : / /

→ The boundary between the agent and the environment is determined by what agent can control anything beyond the agent's control is part of env

- Reward computation to be external of an agent because it defines the task facing the agent and thus must be beyond its ability to change arbitrarily.

→ MDP framework reduces the problem of learning goal-directed behavior to 3 signals:
- the actions chosen by agent
- states that inform these choices
- rewards that define agent's goal

Eg Pick and Place Robot:

- RL to control motor of a robot arm
- Learning agent will have to control the motors directly and have low latency info abt current pos and velocities of mechanical linkages.
- Action – voltage applied to each motor at each joint
- State – latest readings of joint angles and velocities
- Reward – +1 for each obj picked up successfully and placed
- To encourage smooth movement, -ve reward can be given as a function of the moment-to-moment "jerkiness"

Note The action depends on the state it is in.

The transition probs labelling the arrows leaving a action node always sum to 1

Read Eg 3.3 (Recycling Robots) Pg - 52-53

Goals and Rewards :-

→ The purpose or goal of the agent is formalized in terms of a special signal called reward, passing from env to agent

→ At each time step the reward is a simple number $R_t \in \mathbb{R}$

→ Agent's obj: Maximize total amt of reward it receives over time, focusing not just on immediate rewards but on cumulative rewards in long run.

→ Reward Hypothesis : Suggests that all goals and purposes can be effectively represented as the maximization of the expected value of the cumulative sum of received rewards

→ The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.

→ The reward signals can be tailored to represent different goals and objectives in RL framework

Returns and Episodes :-

→ Agent's goal : Maximize reward it receives in long run.

This is formally defined in terms of the expected return, where the return (G_t) is a function of the sequence of rewards received after this step t.

PAGE EDGE
Date : / /

Return = Sum of rewards

Return → $G_t = R_{t+1} + R_{t+2} \cdots R_T$

where T - final time step

- This formulation is suitable for episodic tasks, where the agent - env interaction naturally breaks into subsequences or episodes, such as plays of game or any sort of repeated interaction.

- Each episode ends in a terminal state, followed by a reset to a standard starting state

- Episodes are considered to end in the same terminal state with different rewards for different outcomes. Tasks with episodes of this kind - episodic tasks

- Set of all non terminal state - S
  Set of all states plus terminal state - S^+

- T - random variable that can vary from episode to episode

→ However, for continuing tasks, where the interaction does not break into identifiable episodes at goes on continually without limit, the return formulation is problematic because the step, $T = \infty$ and the return itself could be ∞

- To address this, the concept of discounting is introduced

- Discounting involves weighting rewards received in the future less than immediate rewards, which ensures that the sum of

the discounted rewards remains finite even in continuing tasks

The agent then seeks to maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where γ - a discount rate $(0 \leq \gamma < 1)$

- γ determines the present value of future rewards: a reward received k time steps in future is worth only γ^{k-1} times what it would worth if it were received immediately.
- If $\gamma < 1$, the ∞ sum has a finite value as long as reward sequence $\{R_k\}$ is bounded
- $\gamma = 0$, the agent - myopic - only concerned with immediate rewards - objective to learn how to choose A_t so as to only maximize R_{t+1}.
- γ - reflects tradeoff b/w immediate & future rewards, with a denser γ valuing immediate rewards more heavily.

This formulation allows the agent to make decisions that balance short term gains with long term benefits, even in tasks that continue indefinitely.

→ $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$
$$= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots)$$
$$= R_{t+1} + \gamma G_{t+1}$$

$$\boxed{G_t = R_{t+1} + \gamma G_{t+1}}$$

PAGE EDGE
Date : / /

Note: Return is a sum of an ∞ no of term, it is still finite if reward is non zero and const if $\gamma < 1$

For Eg, if reward is a const +1, then return is

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

Unified Notation for Episodic and Continuing Tasks :-

→ Tasks can be categorized into episodic tasks, where the agent - env interaction breaks down into separate episodes and continuous tasks where interaction does not break into distinct episodes

Episodic tasks are simpler because each action affects only a finite num of subsequent rewards within the episode

→ For episodic tasks, time steps are numbered starting anew from 0 for each episode

∴ $S_{t,i}$ - state representation at time t of episode i

Similarly $A_{t,i}$, $R_{t,i}$, T_i, $\Pi_{t,i}$ etc

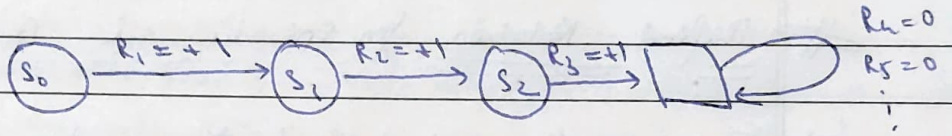
However in practice, the explicit reference to the episode number is often omitted when discussing episodic tasks, as the focus is usually on a single episode or statements that apply to all episodes.

That is we write S_t to refer to $S_{t,i}$

→ To unify the notation for episodic and continuing tasks return is defined as a sum over an infinite number of terms, even for episodic tasks.

This is achieved by considering episode terminates as entering a special absorbing state that transitions only to itself and generates only rewards of 0.

Eg ⟹



- Square box – represents the special absorbing state corresponding to the end of an episode.
- Starting from S_0, we get reward seq $+1, +1, +1, 0, 0, 0 \ldots$
- Summing these, we get same return whether we sum over the first T rewards (here $T=3$) or over full infinite seq.
- This remains true even if we introduce discounting.

∴ The return can be defined using the same eq^n for both types of tasks, with the understanding that the sum remains finite for episodic tasks due to absorbing state.

The return for both episodic and continuing tasks can be written as:

$$G_t = \sum_{k=t+1} \gamma^{k-t-1} R_k,$$

Where T – final time step for episodic tasks (which may be ∞ for continuing tasks)
γ – discount factor
$T = \infty$ or $\gamma = 1$ (but not both)

PAGE EDGE
Date : / /

- This convention simplifies the notation & highlights the parallels b/w episodic and continuing tasks in RL

Policies and Value Functions :-

→ RL often involve estimating value functions, which are functions of states or state-action pairs that estimate how good it is for agent to be in a given state or perform a given action in a given state.

- "Goodness" - defined in term of the expected future or rewards or the the expected return

→ value functions are defined wrt specific ways of acting - policies.

→ Policy - mapping from states to probabilities of selecting each possible action.

- If the agent is following policy π at time t, then $\pi(a|s)$ is the prob that the action A_t is a if the state S_t is s.

- RL's methods specify how the agent's policy is changed as a result of its experience.

→ value function of a state s under a policy π, denoted by $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) \doteq E_\pi\left[G_t \mid S_t = s\right]$$

$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \quad \text{for all } s \in S$$

where E_π — expected return of a random variable given that the agent follows policy π and t is any time step

v_π — state value function for policy π

Note: value of the terminal state = 0

→ Similarly, value of taking action a in state s under a policy π, denoted by $q_\pi(s, a)$ as the expected return starting from s, taking action a and thereafter following policy π :

$$q_\pi(s,a) = E_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

$$= E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

q_π — action value function for policy π

→ Value functions q_π and v_π can be estimated from experience

For Eg: if an agent follows policy π and maintains an avg, for each state encountered of the actual returns that have followed that state, then the avg will converge to the state's value, $v_\pi(s)$, as the no of times that state is encountered approaches ∞.

Similarly keeping separate avg for each action taken in each

PAGE EDG3
Date : / /

state, then these averages will similarly converge to the action values $q_\pi(s,a)$

These estimation methods – Monte Carlo methods – they involve averaging over many random samples of actual returns

* → Value function satisfy recursive relationships known as Bellman eq^.

For any policy π and any state s, the Bellman eq^ for $v_\pi(s)$ is :

$$v_\pi(s) \doteq E[G_t \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

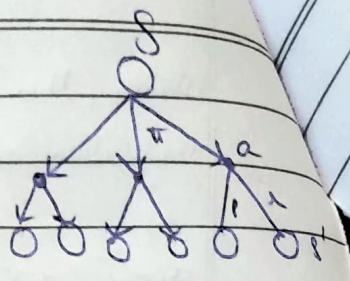
$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']]$$

$$\boxed{v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]}, \text{ for all } s \in S$$

where action a taken from set $A(s)$
next state s' from set S
rewards r from set R

* Bellman eq^ expresses a relationship between the value of a state and the values of its successor state.

- Each open circle - State
  Solid circle - state-action pair



Starting from state s, the root node at top, the agent could take any of some set of actions - 3 (from diagram) based on policy π. From each of these, the env could respond with one of several next states s' along with a reward r, depending on dynamics given by p.
Bellman eq^n avg over all possibilities weighting each by its prob of occuring.

It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward along the way. (expected)

v_π - unique sol^n to its Bellman eq^n

See the eq from TB Eq-60-61

Optimal Policies and Optimal value Functions:

→ Solving a RL task means, finding a policy that achieves a lot of reward over the long run.

→ For finite MDP, we define an optimal policy (π^*) that is better than or equal to all other policies.
This means that for all states s, the expected return is greater than or equal to that of π' for all states - optimal p

In other words, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ policies

PAGE EDGE
Date : / /

- Although there may be more than one, we denote all optimal policies by π_*.
They share same state value function called optimal state value function (v_*):

$$v_*(s) = \max_{\pi} v_\pi(s) \quad \text{for all } s \in S$$

→ Optimal policies also share same optimal action value function (q_*):

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

for all $s \in S$ and $a \in A(s)$

→ For the state action pair (s,a) this function gives the expected return for taking action a in state s and thereafter following an optimal policy.
Thus, q_* in term of v_*:

$$q_*(s,a) = E\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a \right]$$

→ Because v^* - value function for a policy, it must satisfy the self-consistency condition given by the Bellman eq^n for state values

The Bellman optimality eq^n for v_* expresses the fact that value of a state under an optimal policy must equal the expected return for the best action from that state:

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a)$$

$$= \max_a \mathbb{E}_{\pi_*}\left[G_t \,\middle|\, S_t = s, A_t = a\right]$$

$$= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s, A_t = a\right]$$

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right]$$

$$\rightarrow v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right]$$

Bellman optimality eq^n

→ Similarly, Bellman optimality eq^ for q_* is

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, \begin{matrix} S_t = s, \\ A_t = a \end{matrix}\right]$$

$$= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$$

→ For finite MDP, the Bellman optimality eq^ for v_π has a unique sol^ independent of policy.

The Bellman optimality eq^ — actually a sys of eq^, one for each state, so if there are n states, then there are n equations in n unknowns.

If the dynamics p of the env are known, then in principle one can solve the sys of eq^ for v_* using any of a variety of methods for solving sys of non linear eq^.

PAGE EDGE
Date : / /

→ Once we have the optimal value function v^* or optimal action value function q^*, we can easily determine an optimal policy π^* by selecting actions that maximize these functions.
For any state s, an action a is optimal if it achieves the max in the Bellman optimality eq^n

→ Beauty of v_* (optimal state-value) lies in its ability to simplify the decision making process.
v^* is used to evaluate the short term consequences of actions, you can make decisions based on immediate info while ensuring long term optimality. This is because v^* incorporates the reward consequences of all possible future behavior.

→ Having q^* (optimal action-value) simplifies the process of choosing optimal actions even further than using the v^*.
With q^*, the agent does not need to perform a one step ahead search to determine the best action. Instead, for any given state s, the agent can directly select an action that maximizes $q^*(s,a)$.

By using q^*, the agent trades off the simplicity of only needing to represent a function of states (as in v^*) for the convenience of being able to select optimal actions directly, without further computation. This makes q^* a powerful tool for decision making in RL.

→ In practice finding the exact sol^n to these eq^n can be challenging due to computational complexity and the need for complete knowledge of the env's dynamics. ∴ RL algo

often aim to find approximate sol^ to these eq^ using techniques like dynamic programming, Monte Carlo methods and temporal difference learning.

Optimality and Approximation :-

→ In general learning an optimal task is a rare occurrence due to extreme computational cost associated with generating optimal policies for complex tasks

→ Even with a complete & accurate model of the env's dynamics, it is usually not possible to simply compute an optimal policy by solving the bellmans optimality eq^.
For Eg: boards games like chess are a tiny fraction of human experience, yet large, custom designed computers still can't compute the optimal moves. A critical aspect of the problem facing the agent is always the computational power available to it, in particular, the amt of computation it can perform in a single time step

→ Computational power & memory are critical constraints for RL agents.
A large amt of memory is often required to build up approximations of value functions, policies and models.
In cases with small, finite state sets — tabular methods stores values in arrays or table for each state or state-action pair.
However, in many practical situations, the no of states far exceeds what can be feasibly stored in a table, necessitating the use of more compact parameterized function representations to approximate the functions.

PAGE EDGE
Date : / /

→ RL inherently involves settling for approx due to these constraints.
However, it also presents us with some unique opportunities for achieving useful approximations.

For Eg: an agent might encounter many states with such low probs that selecting suboptimal optimal actions for them has little impact on the amt of reward the agent receives.

This allows RL to focus more on effort on learning good decisions for frequently encountered states while expending less effort on infrequently encountered states.
This selective focus on important states is a key prop that distinguishes RL from other methods for approximately solving MDP.

Read Summary
Pg. 68