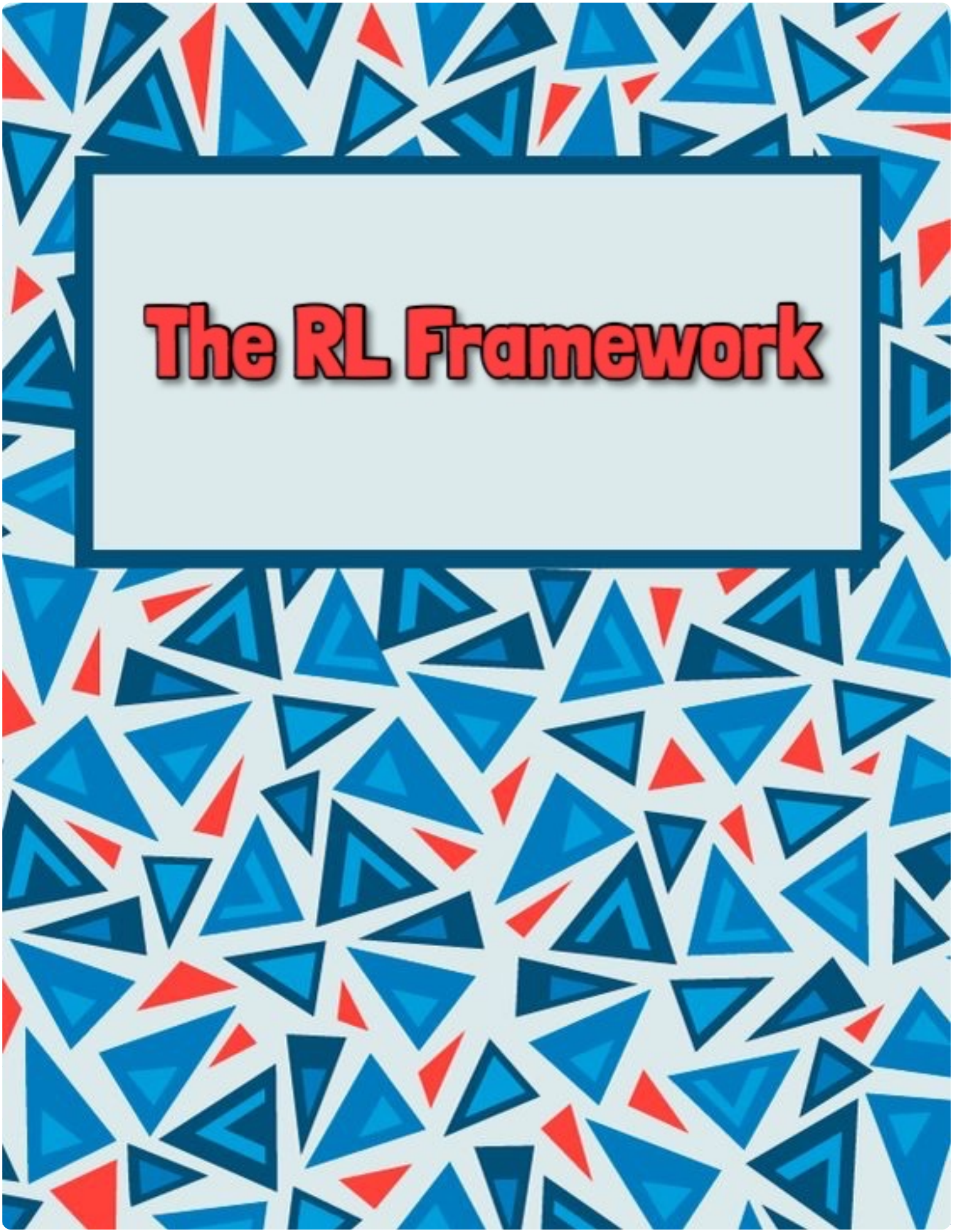
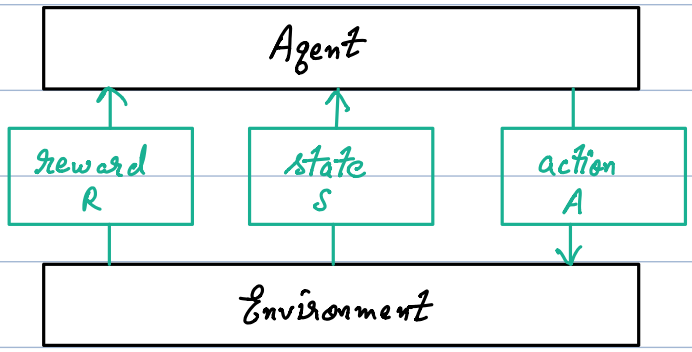

The RL Framework



Initially, The environment is in state (S_0)

→ The Agent performs an action (A_0)

→ In response to A_0, the environment state changes to (S_1) & also a reward (R_1) is returned.

State - Action - Reward Sequence :- $S_0\ A_0\ R_1\ S_1\ A_1\ R_2\ S_2\ A_2\ R_3$ .....

Goal of the Agent: Maximize expected cumulative reward i.e. the sum of the rewards of all timesteps.

→ **Episodic Task**: Interaction ends at time step 'T'. ($S_0\ A_0\ R_1\ S_1\ A_1 \dots R_T\ S_T$)

→ **Continuing Task**: Interaction continues without limit. ($S_0\ A_0\ R_1\ S_1\ A_1$ .....)

Reward Hypothesis:- All goals can be framed as the maximization of "expected" cumulative reward.

At any timestep 't', the reward returned is:-

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$ (Sum of current & future rewards)

It is hard to predict all the future rewards with certainity & hence we choose the maximize the [Expected] G_t at timestep 't'.

Discounted Reward (γ) :-

To regulate how much we should care about long term rewards, we use a parameter gamma (γ) as the discounting factor. Gamma is choosen by the developer & not learned.

Now,

$$\text{Discounted Reward } (G_t) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

$$= \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \quad \text{where } \gamma \in [0, 1]$$

if $\gamma = 1$, this results in undiscounted reward i.e. all future rewards are important.

if $\gamma = 0$, $G_t = R_{t+1} \Rightarrow$ This means that we only care about the current reward.

The more the value of γ, the more we care about distant rewards in the future.

In this classic reinforcement learning task, a cart is positioned on a frictionless track, and a pole is attached to the top of the cart. The objective is to keep the pole from falling over by moving the cart either left or right, and without falling off the track.

In the OpenAI Gym implementation, the agent applies a force of +1 or -1 to the cart at every time step. It is formulated as an episodic task, where the episode ends when (1) the pole falls more than 20.9 degrees from vertical, (2) the cart moves more than 2.4 units from the center of the track, or (3) when more than 200 time steps have elapsed. The agent receives a reward of +1 for every time step, including the final step of the episode. You can read more about this environment in OpenAI's github.

QUESTION 2 OF 3

Say that the reward signal is amended to only give reward to the agent at the end of an episode. So, the reward is 0 for every time step, with the exception of the final time step. When the episode terminates, the agent receives a reward of **-1**. Which discount rates would encourage the agent to keep the pole balanced for as long as possible? (Select all that apply.)

- [ ] The discount rate is 1.
- [x] The discount rate is 0.9.
- [x] The discount rate is 0.5.
- [ ] (None of these discount rates would help the agent, and there is a problem with the reward signal.)

Correct!

Without discounting, the agent will always receive a reward of -1 (no matter what actions it chooses during the episode), and so the reward signal will not provide any useful feedback to the agent.

With discounting, the agent will try to keep the pole balanced for as long as possible, as this will result in a return that is relatively less negative.

QUESTION 3 OF 3

Say that the reward signal is amended to only give reward to the agent at the end of an episode. So, the reward is 0 for every time step, with the exception of the final time step. When the episode terminates, the agent receives a reward of **+1**. Which discount rates would encourage the agent to keep the pole balanced for as long as possible? (Select all that apply.)

- ☐ The discount rate is 1.
- ☐ The discount rate is 0.9.
- ☐ The discount rate is 0.5.
- ☑ (None of these discount rates would help the agent, and there is a problem with the reward signal.)

If the discount rate is 1, the agent will always receive a reward of +1 (no matter what actions it chooses during the episode), and so the reward signal will not provide any useful feedback to the agent.

If the discount rate is 0.5 or 0.9, the agent will try to terminate the episode as soon as possible (by either dropping the pole quickly or moving off the edge of the track).

Thus, you are correct - we must redesign the reward signal!

RL as a (finite) Markov Decision Process :-

$S \rightarrow$ set of all non-terminal states.

(In episodic tasks, $S^+ \rightarrow$ all states including terminal states)

$A \rightarrow$ set of all possible actions available to the agent.

(If a state has only a subset of actions available use $A(s)$ denoting set of actions available in state $s \in S$.)

$R \rightarrow$ set of rewards

⇒ The one-step dynamics of the environment :

$$p(s', r \mid s, a) = P(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a) \quad \forall\, s, s', a, r$$

⇒ Discount rate $(\gamma) \Rightarrow [0, 1]$ (In general a value close to 1, to avoid short sightedness of the agent.)

⇒ In finite MDP's, S / S^+ & A are finite sets.

For eg. "CartPole-v0" is not a finite MDP because it has infinite state space due to real coordinate values.

Policies (π):-

These are mappings from the set of environment states (S) to the Actions (A) i.e. $\pi : S \rightarrow A$

There are 2 types of policies:-

(i) Deterministic Policy :- Mapping $\pi : S \rightarrow A$, each state has a corresponding fixed allowable action.

Eg. π (low-state) = recharge, π (high-state) = search

(ii) Stochastic Policy :- Mapping $\pi : S \times A \rightarrow [0,1]$, given a state ($s$) & action ($a$) it returns the probability that the agent takes action (a) while in state (s)

$$\pi(a|s) = P(A_t = a \mid S_t = s)$$

Eg. π (recharge | low) = 0.5 π (search | high) = 0.9

π (wait | low) = 0.4 π (wait | high) = 0.1

π (search | low) = 0.1

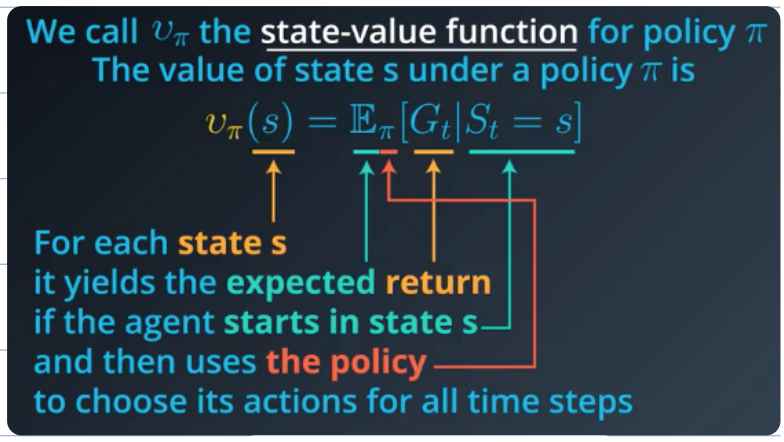
Deterministic policy can be represented using stochastic notation.

Eg. π (recharge | low) = 1, π (search | low) = 0

State-Value Function (v_π):-

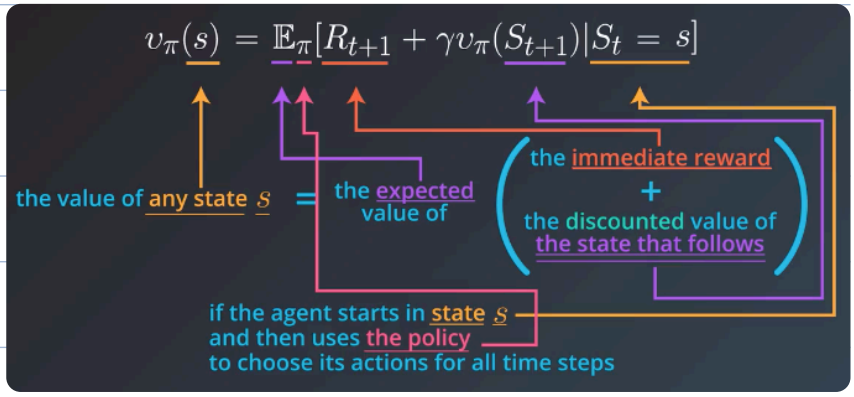
Expected reward generated by following the policy (π) from state (S) is given by the state-value function $v_\pi(s)$.

It estimates how good it is for the agent to be present in state (S) in terms of future expected return / reward.



$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right]$$

> Bellman Expectation Equation :-



For any state (s) instead of calculating $v_\pi(s)$ from scratch, we can use the next state $v_\pi(s+1)$ to recursively compute $v_\pi(s)$.

i.e. value of current state depends on the value of all possible successor states.

→ For deterministic policy :-

Probability of s', r given s with action $\pi(s)$

$$v_\pi(s) = \sum_{s' \in S, r \in R} p(s', r \mid s, \pi(s)) \, (r + \gamma \, v_\pi(s'))$$

- $\sum_{s' \in S, r \in R}$: Sum over all possibilities
- s : current state
- $\pi(s)$: Action performed
- $v_\pi(s')$: next state-value function

→ For stochastic policy :-

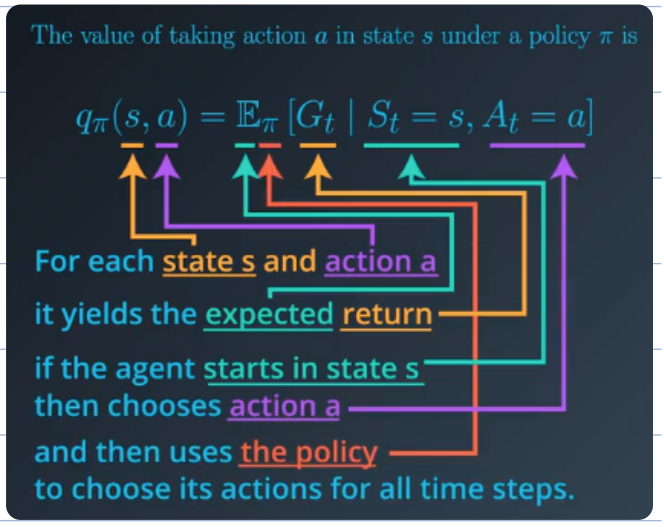
$$v_\pi(s) = \sum_{s' \in S, r \in R} \pi(a \mid s) \, p(s', r \mid s, a) \, (r + \gamma \, v_\pi(s'))$$

- $\pi(a \mid s)$: probability of selecting action (a) in state (s)

⊛ Value of terminal state is always zero (0).

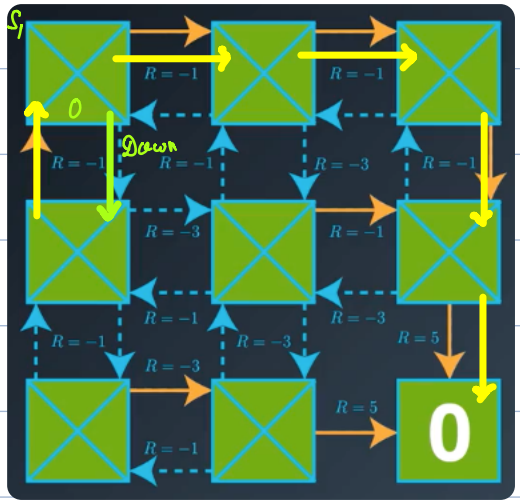
Action-Value Function (q_π) :-

For each state & action, q_π yields the expected return if the agent starts in that state (s), takes action (a), and then follows the policy for all the future time steps.



Eg. For state (S_1), action (Down):-

$$q_\pi(S_1, \text{Down}) = \underbrace{(-1)}_{\text{reward from action "down" on } S_1} + \underbrace{(-1) + (-1) + (-1) + (-1) + 5}_{\text{reward from following the policy for all future timesteps}} = 0$$



Orange arrows = optimal policy

For a deterministic policy (π),

$$v_\pi(s) = q_\pi(s, \pi(s)) \quad \forall s \in S$$

Optimal Policy:-

$\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in S$

(Note:- It is often possible to find two policies that cannot be compared.)

An optimal policy (π_*) satisfies $\pi_* \geq \pi$ for all π

(Note:- Optimal policy is guaranteed to exist, but may not be unique.)

The corresponding optimal state-value function is denoted by v_*

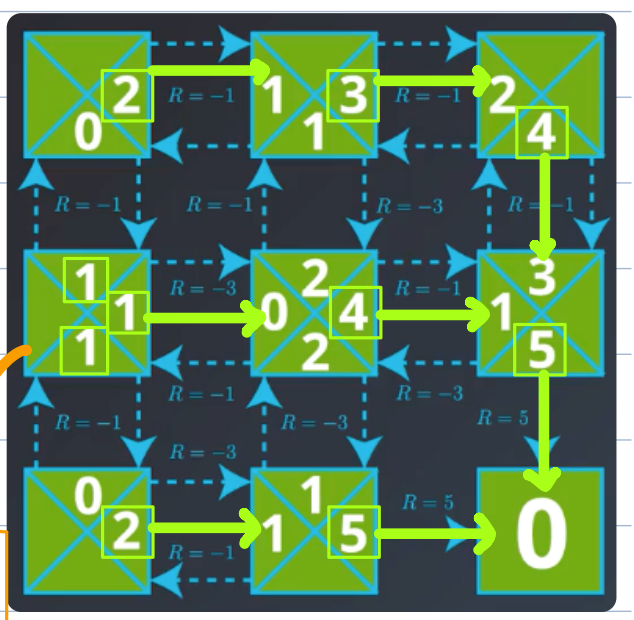
optimal action-value function is denoted by q_*

:> How are optimal policies found?

Interaction $\longrightarrow q_* \longrightarrow \pi_*$

The agent interacts with the environment to find the optimal action-value function (q_*) which is then used to find the optimal policy by taking the maximum reward action in each state.

$$\pi_*(s) = \underset{a \in A(s)}{\text{argmax}}\ q_*(s, a) \quad \forall s \in S$$



In this state, all 3 actions are equally rewarding, hence, any of them will result in optimal policy.

Eg.

q_*

	a_1	a_2	a_3
s_1	1	(2)	-3
s_2	-2	1	(3)
s_3	(4)	(4)	-5

Thus, the optimal policy π_* for the corresponding MDP must satisfy:

- $\pi_*(s_1) = a_2$ (or, equivalently, $\pi_*(a_2|s_1) = 1$), and
- $\pi_*(s_2) = a_3$ (or, equivalently, $\pi_*(a_3|s_2) = 1$).

This is because $a_2 = \arg\max_{a \in \mathcal{A}(s_1)} q_*(s_1, a)$, and $a_3 = \arg\max_{a \in \mathcal{A}(s_2)} q_*(s_2, a)$.

In other words, under the optimal policy, the agent must choose action a_2 when in state s_1, and it will choose action a_3 when in state s_2.

As for state s_3, note that $a_1, a_2 \in \arg\max_{a \in \mathcal{A}(s_3)} q_*(s_3, a)$. Thus, the agent can choose either action a_1 or a_2 under the optimal policy, but it can never choose action a_3. That is, the optimal policy π_* must satisfy:

- $\pi_*(a_1|s_3) = p$,
- $\pi_*(a_2|s_3) = q$, and
- $\pi_*(a_3|s_3) = 0$,

where $p, q \geq 0$, and $p + q = 1$.

Note :- For optimal policy (π_*) :-

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