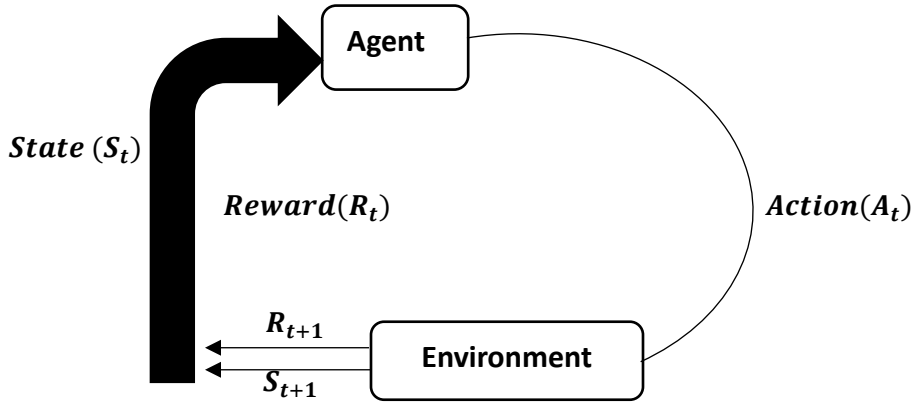
Reinforcement Learning (RL) [Assignment (1) – GROUP-2]

1. What it is and how it works (Brief Formal Statement)

Reinforcement learning is the area of machine learning where an agent learns to make sequential decisions by interacting with an environment to maximize cumulative reward. Formally the interaction is modelled as a Markov Decision Process **(MDP)** where, at discrete time steps (t) the agent is in state (S_t) takes action (A_t) receives scalar reward (R_{t+1}), and transitions to next state (S_{t+1}). The agent's objective is to learn a policy $\pi(a|s)$ that maximizes the expected discounted return $(G_t) = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, where\ 0 \leq \gamma \leq 1$ is the discount factor. This formulation and the foundations are standard according to Sutton and Barto.

Figure 1: Simple diagram illustrating an agent-environment loop.



Key quantities:

1. State - value $(V^{\pi}(s) = E_{\pi}[G_t|S_t = s]$.
2. Action – value (Q-Value): $Q^{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a]$.
   Bellman relation for optimal action - value (Q^*) (Bellman Optimality)

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

2. Monte Carlo methods (Model – Free Monte Carlo)

Monte Carlo (MC) methods in RL estimate value functions from complete episodes of experience (no model of the environment is required). They are model – free and rely on averaging actual returns observed after state visits.

2.1 Definitions and update rule (every – visit MC for action-values). For an episode, for each time step (t) we can compute the return

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}.$$

If we estimate $\boldsymbol{Q(s,a)}$ as the sample average of returns following all occurrences of $(\boldsymbol{s},\boldsymbol{a})$, then after observing the "nth" return $\boldsymbol{G^{(n)}}$ the incremental average update is:

Step 1 (incremental average): if current estimate after $(\boldsymbol{n-1})$ samples is a $(\boldsymbol{Q_n - 1})$, on seeing $\boldsymbol{G^{(n)}}$ do

$$\boldsymbol{Q_n = Q_{n-1} + \frac{1}{n}\left(G^{(n)} - Q_{n-1}\right)}.$$

This avoids storing all past returns and is mathematically identical to averaging.

2.2 Worked arithmetic example (digit – by - digit):
Suppose you have three episode returns observed for a state-action pair: **5, 7, 6** (discount already applied or $(\boldsymbol{\gamma = 1})$ for simplicity). Compute their sample average step by step.

Step 1: We will add the first two: **(5 + 7) = 12.**
Step 2: We then add the third: **(12 + 6) = 18.**
Step 3: We then divide by **3:** $\frac{18}{3} = \mathbf{6}$. So, the Monte Carlo estimate is 6.

Using the incremental formula:
Start $\boldsymbol{Q_0}$ undefined, after first return **(n = 1):** $\boldsymbol{Q_1 = 5}$.
After second return **(n=2):** $\boldsymbol{Q_2 = Q_1 + \frac{1}{2}(7 - Q_1) = 5 + 0.5(7-5) = 5 + 0.5 \times 2 = 6}$.
After third return **(n=3):** $\boldsymbol{Q_3 = 6 + \frac{1}{3}(6 - 6) = 6}$.

2.3 Properties and when to use MC
MC methods converge (with enough samples) to correct values under standard assumptions, but they require episodes that terminate (or very long episodes) and can be high – variance.

3. Model-Based Monte Carlo (What it means & how it operates)

"Model – Based Monte Carlo" refers to using an estimate model of the environment (estimated transition probabilities and reward function) to run simulated rollouts (Monte Carlo Simulations) and evaluate policies or estimate returns. The approach has two phases: learn a model from data, then perform planning by sampling trajectories from that learned model.

3.1 Steps (conceptually):
Step 1: Collect transitions $(\boldsymbol{s},\boldsymbol{a},\ \boldsymbol{r},\ \boldsymbol{s'})$ from interaction.
Step 2. Estimate model: for each $(\boldsymbol{s},\boldsymbol{a})$ form empirical transition probabilities $\boldsymbol{\hat{P}(s'|s,a) = 3/10}$.
Step 3. Use the learned model to simulate many episodes (rollouts) under a policy $\boldsymbol{\pi}$ and estimate $\boldsymbol{V^{\pi}}$ *or* $\boldsymbol{Q^{\pi}}$ by Monte Carlo averaging of simulated returns. Optionally use dynamic programming on the estimated model.

3.2 Small numeric example of estimating a transition probability:
Observed transitions from $(\boldsymbol{s},\boldsymbol{a})$: $\boldsymbol{s}1$ occurred **4** times, $\boldsymbol{s}2$ occurred **6** times (total 10).
Then $\boldsymbol{\hat{P}(s1|s,a) = \frac{4}{10} = 0.4}$ *and* $\boldsymbol{\hat{P}(s2|s,a) = \frac{6}{10} = 0.6}$. if observed rewards from $(\boldsymbol{s},\boldsymbol{a})$ were {1,0,1,1,01,1,0,1,1} then sum = count success: step-by-step sum will be;
(1+0=1, +1=2, +1=3, +1 =4, +11=5, +1=6, +1=**7**), implying, Total is 7 across **10**, so $\boldsymbol{\hat{R}(s,a) = \frac{7}{10} = 0.7}$.

3.3 Advantages/Disadvantages
Model-based MC can be more sample-efficient because once a reasonably accurate model is learned, planning can reuse it. But learning an accurate model can be hard in high – dimensional or continuous spaces.

4. Temporal – Difference (TD) control: SARSA (on-policy) & Q-Learning (off-policy)

Temporal – Difference methods update estimates using one – step observed transitions rather than waiting for episode termination. Two canonical algorithms for control are **SARSA** (on – policy TD control) and Q-learning (off-policy TD control). Their update equations are closely related but differ in whether the update target uses the action actually taken (on-policy) or the greed action (off-policy).

4.1 **SARSA** (State-Action-Reward-State-Action)

Update rule (Standard form):

HE $\boldsymbol{Q(S_t, A_t) \leftarrow\ Q(S_t, A_t) +\ \alpha(R_{t+1} +\ \gamma Q(S_{t+1}, A_{t+1}) -\ Q(S_t, A_t))}$.

This is on – policy: the next action $\boldsymbol{A_{t+1}}$ is drawn from the current policy (possibly exploratory). SARSA was introduced in the technical note by Rummery & Niranjan and popularized in Sutton & Barto.

Worked numeric example (digit-by-digit):

Assume $\boldsymbol{Q(S_t, A_t) = 2.50}$, learning rate $\boldsymbol{\alpha = 0.10}$, reward $\boldsymbol{R_{t+1} = 1.00,}$ discount $\boldsymbol{\gamma = 0.90}$, and $\boldsymbol{Q(S_{t+1}, A_{t+1}) = 2.80}$.

Step1: Compute target $\boldsymbol{T = R_{t+1} +\ \gamma Q(S_{t+1}, A_{t+1})}$.

$\boldsymbol{Y} \times\ \boldsymbol{\varrho}$ next $= 0.90 \times 2.80 =$ (0.9 2.8). Compute: (2.8 $\times$ 0.9 = 2.52).

$$Then\ \boldsymbol{T} = 1.00 + 2.52 = 3.52.$$

Step 2: TD error = $\boldsymbol{T - Q(S_t, A_t) = 3.52 - 2,50 = 1.02}$

Step 3: Update $\boldsymbol{Q_{new} = 2.50 + 0.10\ \times 1.02 = 2.50 + 0.102 = 2.602}$.

So, **Q** becomes 2.602.

4.2 **Q-Learning (off – policy)**

$\boldsymbol{Q(S_t, A_t) \leftarrow\ Q(S_t, A_t) +\ \alpha(R_{t+1} +\ \gamma \max_a Q\ (S_{t+1}, a) -\ Q(S_t, A_t))}$.

Q-Learning aims to learn the optimal action-value $\boldsymbol{Q^*}$ and is off-policy because the updated target uses the greedy action $\boldsymbol{max_a Q}$ regardless of the action actually taken. Watkins & Dayan proved convergency under standard assumptions (exploration ensuring all state-action pairs are visited infinitely often, decaying learning rate, etc.).

Worked numeric example (digit-by-digit):

We will same numbers except replace $\boldsymbol{Q(S_{t+1}, A_{t+1})}$ with $\boldsymbol{max_a Q(S_{t+1}, a)}$ **= 3.00.**

Step 1: $\boldsymbol{Y \times\ max = 0.9\ \times\ 3.00\ = 2.70.}$
Step 2: target $\boldsymbol{R + Y\ max = 1.00 + 2.70}$.
Step 3: TD error = **3.70 – 2.50 = 1.20.**
Step 4: Update Q new = $\boldsymbol{2.50 + 0.10 \times 1.20 = 2.50 + 0.12 = 2.62}$.

So, **Q** is **2.62** after the update.

4.3 Quick Comparison (Statement)

5. Multi-Armed Bandits (MABs) & Bayesian bandits

6. Pseudocode summaries (short and precise)

7. Practical considerations & recommendations (implementation, hyperparameters)

8. Overall Summaries & Notations.

9. Appendix – Abbreviation & Sort meanings.