22CM604

Hall Ticket Number:

III/VI B.Tech (Regular/Supplementary) DEGREE EXAMINATION

May, 2026 — **Artificial Intelligence & Machine Learning**

Sixth Semester — **Reinforcement Learning**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsorily. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14 = 56 Marks)**

			CO	BL	M
1	a)	What is Reinforcement Learning?	CO1	L1	1M
	b)	What is meant by exploration and exploitation?	CO1	L2	1M
	c)	Define policy in RL	CO1	L1	1M
	d)	List the components of an MDP	CO2	L1	1M
	e)	Define the state value function	CO2	L1	1M
	f)	What is the purpose of the discount factor?	CO2	L2	1M
	g)	What is model-based reinforcement learning?	CO2	L2	1M
	h)	Expand SARSA	CO3	L1	1M
	i)	Write the incremental update rule in the Monte Carlo methods	CO3	L3	1M
	j)	What is the difference between on-policy and off-policy learning?	CO3	L2	1M
	k)	Define TD (0) prediction	CO3	L1	1M
	l)	What are eligibility traces in reinforcement learning?	CO4	L2	1M
	m)	What is a TD error (δ_t)?	CO4	L1	1M
	n)	Define a target network in DQN.	CO4	L2	1M
		Unit-I			
2	a)	Explain different applications of Reinforcement Learning.	CO1	L2	7M
	b)	How does the SoftMax exploration approach work in the context of smart traffic light control? Explain the steps and calculate the probability for each action when Q(a1)=5, Q(a2)=25, Q(a3)=15, and τ=3**.**	CO1	L3	7M
		(OR)			
3	a)	Describe the elements of Reinforcement Learning in detail	CO1	L2	7M
	b)	Consider a casino with three slot machines (bandits): Machine 1 (M1) pays out with probability 0.3, Machine 2 (M2) pays out with probability 0.9, and Machine 3 (M3) pays out with probability 0.6. You want to maximise your total reward over 8 rounds using the ε-greedy algorithm with ε = 0.2.	CO1	L4	7M
		Unit-II			
4	a)	Derive the Bellman equation for the state value function	CO2	L4	7M
	b)	Compare the policy iteration and value iteration	CO2	L4	7M
		(OR)			
5	a)	Explain the complete structure of MDP with an example (Grid World example)	CO2	L2	7M
	b)	Explain value iteration with an algorithm.	CO2	L3	7M
		Unit-III			
6	a)	Differentiate between SARSA and Q-learning	CO3	L4	7M
	b)	Explain the Monte Carlo prediction with an example	CO3	L3	7M
		(OR)			
7	a)	Discuss the advantages of TD learning methods	CO3	L2	7M
	b)	Derive the equation for the combination of MC error and TD error	CO3	L4	7M
		Unit-IV			
8	a)	Explain the backward view of the TD(λ) Algorithm with an example problem	CO4	L3	7M
	b)	Explain the n-step TD prediction Algorithm with an example. Estimate the **value function V(s)** for a given policy using **n-step TD**.	CO4	L4	7M

States: A → B → C → Terminal

Rewards: $R_1 = 0, R_2 = 0, R_3 = 1$

$\gamma = 1, \alpha = 0.1, n = 2$

(OR)

9	a)	Explain the **architecture of DQN** with a diagram	CO4	L2	7M
	b)	Explain the forward view of the TD(λ) Algorithm with an example problem	CO4	L3	7M

III/VI B.Tech (Regular/Supplementary) DEGREE EXAMINATION

May, 2026 — **Artificial Intelligence & Machine Learning**

Sixth Semester — **Reinforcement Learning**

Time: Three Hours — **Maximum:** 70 Marks

SCHEME OF EVALUATION

Answer question 1 compulsorily. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14 = 56 Marks)**

1	**a)**	**What is Reinforcement Learning?**	**CO1**	**L1**	**1M**
		Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns to make decisions by interacting with an environment and receiving rewards or penalties for its actions			
	b)	**What is meant by exploration and exploitation?**	**CO1**	**L2**	**1M**
		Exploration means trying out new actions to gather more information about the environment. Exploitation means choosing the action that gives the Highest Reward based on what the agent already knows.			
	c)	**Define policy in RL.** A policy (π) is a strategy used by an agent to select actions in different states. It maps states to actions**.**	**CO1**	**L1**	**1M**
	d)	**List the components of an MDP.** A Markov Decision Process (MDP) consists of: 1.States (S) 2.Actions (A) 3.Transition Probability (P) 4.Reward Function (R) 5.Discount Factor (γ)	**CO2**	**L1**	**1M**
	e)	**Define the state value function.** It is defined as the normal or general reward that the agent gets in future. **$V(s)=E_\pi [G_t \mid S_t=s]$**	**CO2**	**L1**	**1M**
	f)	**What is the purpose of the discount factor?** The discount factor (γ) determines the importance of future rewards relative to immediate rewards. Its value lies between 0 and 1.	**CO2**	**L2**	**1M**
	g)	**What is model-based reinforcement learning?** Model-based RL uses a model of the environment (state transitions and rewards) to plan actions and predict future outcomes.	**CO2**	**L2**	**1M**
	h)	**Expand SARSA.** SARSA = State – Action – Reward – State – Action	**CO3**	**L1**	**1M**
	i)	**Write the incremental update rule in the Monte Carlo methods.** V(St)←V(St)+α[Gt−V(St)] where: $V(S_t)$= current value estimate • G_t= return • α= learning rate	**CO3**	**L3**	**1M**

j)	**What is the difference between on-policy and off-policy learning?** **on-policy:** Learn about the same policy that is being followed (e.g, SARSA). **off-policy:** Learn about a different target policy than the behaviour policy (e.g., Q-Learning).	**CO3**	**L2**	**1M**
k)	**Define TD (0) prediction** TD(0) prediction is a Temporal Difference learning method that updates state values using the immediate reward and the estimated value of the next state. $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$	**CO3**	**L1**	**1M**
l)	**What are eligibility traces in reinforcement learning?** Eligibility trace is a short-term memory in reinforcement learning that tracks how recently and frequently a state or action was visited, helping assign credit for rewards over time.	**CO4**	**L2**	**1M**
m)	**What is a TD error (δ_t)?** The Temporal Difference (TD) error measures the difference between the predicted value and the updated estimate. $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$	**CO4**	**L1**	**1M**
n)	**Define a target network in DQN.** A target network in Deep Q-Networks (DQN) is a separate neural network used to compute target Q-values. Its parameters are updated periodically from the main network to stabilise training.	**CO4**	**L2**	**1M**

Unit-I

2 **a)**	**Explain different applications of Reinforcement Learning.**	**CO1**	**L2**	**7M**

1. **Robotics:**
   - Purpose: Teach robots to perform complex tasks.

Examples: Industrial robots for assembly and manufacturing.

- Autonomous robots for navigation, grasping, and manipulation

2. Autonomous Vehicles:

- Purpose: Train self-driving cars to make real-time driving decisions.

Examples: Lane-keeping, obstacle avoidance, and traffic management.

- Efficient route planning under uncertain traffic conditions.

3. Healthcare :

- Purpose: Improve personalized treatment and resource management.

Examples: Designing personalized drug dosage systems.

- ❖ Training agents for medical diagnosis and robotic surgeries.

4. Game AI

- Purpose: Develop intelligent agents capable of playing games competitively.

Examples:

- AlphaGo (mastering Go).
- OpenAI Five (competing in Dota 2).
- Atari game-playing agents.

5. Financial Systems

- Purpose: Optimize trading strategies and portfolio management.

Examples:

- Algorithmic trading in stock markets.
- Risk management and fraud detection.

6. Smart Cities and IoT:

- Purpose: Optimize resource usage and enhance automation.

Examples:

- Energy management in smart grids
- Traffic signal optimization.

7. Natural Language Processing (NLP):

- Purpose: Improve conversational AI and language understanding.

Examples:

- Training chatbots to handle diverse queries.
- Enhancing machine translation and summarisation.

8. Education:

- Purpose: Personalise learning experiences.

Examples: Adaptive tutoring systems that adjust difficulty levels based on student performance

9. Marketing and Advertising:

- Real-time ad placement, budget allocation, and user engagement optimisation.

10. Logistics and Delivery:

- Route optimisation, warehouse management, and drone delivery path planning.

11. Education Technology:

- Adaptive learning systems that tailor content delivery to student performance and behaviour.

b)	**How does the SoftMax exploration approach work in the context of smart traffic light control? Explain the steps and calculate the probability for each action when Q(a1)=5, Q(a2)=25, Q(a3)=15, and τ=3.**	**CO1**	**L3**	**7M**

SoftMax exploration chooses actions probabilistically rather than always selecting the highest Q-value. Actions with higher Q-values are more likely to be selected, but lower-valued actions still have a chance, encouraging exploration.

SoftMax Formula

$$P(a_i) = \frac{e^{Q(a_i)/\tau}}{\sum_j e^{Q(a_j)/\tau}}$$

where:

- $Q(a_i)$= estimated value of action a_i
- τ= temperature parameter
- Higher $\tau \rightarrow$ more exploration
- Lower $\tau \rightarrow$ more exploitation

Given

- $Q(a_1) = 5$
- $Q(a_2) = 25$
- $Q(a_3) = 15$
- $\tau = 3$

Step 1: Calculate $Q(a_i)/\tau$

$$\frac{Q(a_1)}{\tau} = \frac{5}{3} = 1.667$$
$$\frac{Q(a_2)}{\tau} = \frac{25}{3} = 8.333$$
$$\frac{Q(a_3)}{\tau} = \frac{15}{3} = 5$$

Step 2: Calculate Exponentials

$$e^{8.333} \approx 4160.262$$
$$e^{5} \approx 148.413$$

Step 3: Compute the Denominator

$$5.294 + 4160.262 + 148.413 = 4313.969$$

Step 4: Calculate Probabilities

For a_1:

$$P(a_1) = \frac{5.294}{4313.969} \approx 0.00123$$
$$P(a_1) \approx 0.123\%$$

For a_2:

$$P(a_2) = \frac{4160.262}{4313.969} \approx 0.9644$$
$$P(a_2) \approx 96.44\%$$

For a_3:

$$P(a_3) = \frac{148.413}{4313.969} \approx 0.0344$$
$$P(a_3) \approx 3.44\%$$

Final Probabilities

Action	Q-Value	Probability
a_1	5	0.123%
a_2	25	96.44%
a_3	15	3.44%

Interpretation

For the traffic light controller:

- Action a_2(highest Q-value) is chosen most of the time (96.44%).
- Action $a_3$is occasionally explored (3.44%).
- Action $a_1$is rarely selected (0.123%).

(OR)

3 a) Describe the elements of Reinforcement Learning in detail **CO1 L2 7M**

The elements used in reinforcement learning:

- ❖ Agent
- ❖ Environment
- ❖ State
- ❖ Action
- ❖ Reward
- ❖ Policy
- ❖ Reward function
- ❖ Value function
- ❖ Model of the environment

Agent: The learner or decision maker

Environment: Everything the agent interacts with**.**

State: A specific situation in which the agent finds **itself.**

Action: All possible moves the agent can make.

Reward: Feedback from the environment based on the action taken.

Policy:

- ❖ Defines the agent behaviour at a given time.
- ❖ A mapping from state to actions.

Reward function:

- ❖ Defines the goal in an RL problem.

- ❖ Policy is altered to achieve this goal.

Value function:

- ❖ It is used to perform actions on particular state.
- ❖ Value of a state is the total amount at reward an agent can expert to accumulate over the future starting from that state.

Model of the environment:

- ❖ Predict mimic behaviour of the environment
- ❖ Used for planning and if knows current state and action then predict the resultant next state and next reward.

b) **Consider a casino with three slot machines (bandits): Machine 1 (M1) pays out with probability 0.3, Machine 2 (M2) pays out with probability 0.9, and Machine 3 (M3) pays out with probability 0.6. You want to maximise your total reward over 8 rounds using the ε-greedy algorithm with ε = 0.2.** **CO1 L4 7M**

Multi-Armed Bandit Problem Using ε-Greedy Algorithm

Given:

Machine 1 (M1): Reward probability = 0.3

Machine 2 (M2): Reward probability = 0.9

Machine 3 (M3): Reward probability = 0.6

Number of rounds = 8

ε = 0.2

Step 1: Understanding ε-Greedy

The ε-greedy algorithm balances:

Exploration (probability ε = 0.2): Choose a random machine.

Exploitation (probability 1 − ε = 0.8): Choose the machine with the highest estimated reward.

Thus,

$$P(\text{explore}) = 0.2$$
$$P(\text{exploit}) = 0.8$$

Step 2: Initial Exploration

Assume each machine is tried once in the first three rounds.

Round	Selected Machine	Reward (Sample Outcome)	Estimated Value
1	M1	0	Q(M1)=0
2	M2	1	Q(M2)=1
3	M3	1	Q(M3)=1

Since M2 and M3 have the highest estimates, further exploration and exploitation continue.

Step 3: Expected Number of Explorations and Exploitations

For 8 rounds:

Explorations

$$8 \times 0.2 = 1.6$$

≈ 2 exploration rounds

Exploitations

$$8 \times 0.8 = 6.4$$

≈ 6 exploitation rounds

Step 4: Expected Selection Frequency

During exploration (2 rounds):

Each machine has equal probability:

$$\frac{2}{3} = 0.667$$

Expected selections:

M1 ≈ 0.67

M2 ≈ 0.67

M3 ≈ 0.67

During exploitation (6 rounds):

The algorithm eventually identifies M2 as the best machine because:

$$0.9 > 0.6 > 0.3$$

Hence most exploitation rounds choose M2.

Approximate selections:

Machine	Exploration	Exploitation	Total
M1	0.67	0	0.67
M2	0.67	6	6.67
M3	0.67	0	0.67

Step 5: Expected Reward

Expected reward from each machine:

M1

$$0.67 \times 0.3 = 0.201$$

M2

$$6.67 \times 0.9 = 6.003$$

M3

$$0.67 \times 0.6 = 0.402$$

Total Expected Reward

$$0.201 + 6.003 + 0.402$$
$$= 6.606$$

Final Answer

Using the ε-greedy algorithm (ε = 0.2) over 8 rounds:

About 2 rounds are used for exploration.

About 6 rounds are used for exploitation.

The algorithm gradually identifies Machine 2 (M2) as the optimal machine because it has the highest reward probability (0.9).

Most selections are allocated to M2.

The expected total reward is approximately:

$$\boxed{6.61}$$

Unit-II

4 a) Derive the Bellman equation for the state value function **CO2 L4 7M**

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \qquad (3.12)
\end{aligned}$$

b) Compare the policy iteration and value iteration **CO2 L4 7M**

Aspect	**Policy Iteration**	**Value Iteration**
Main Idea	Alternates between **policy evaluation** and **policy improvement**	Updates value function using **Bellman optimality equation** directly
Steps per Iteration	1. Evaluate current policy fully 2. Improve policy	1. Update value function 2. Derive policy after convergence
Value Update	Evaluates **entire policy** to update value function	Uses a **max over actions** at each step

Policy Evaluation	Full evaluation (can be many iterations per step)	One-step truncated evaluation (single update per state)
Policy Improvement	Separate step after evaluation	Implicit in value update (max over actions)
Computation per Iteration	Higher (due to full policy evaluation)	Lower (only one value update per state per iteration)
Convergence Speed	Fewer iterations, but more costly per iteration	More iterations, but cheaper per iteration
When to Use	Small to medium state spaces	Large state spaces or when evaluation is expensive
Final Output	Optimal policy and value function	Optimal value function first, then derive policy

(OR)

5 a) Explain the complete structure of MDP with an example (Grid World example). **CO2 L2 7M**

Problem Setup:

- **States (S): 9 states numbered 1 to 9.**
- **Actions (A): Up, Down, Left, Right.**
- **Reward (r): -1 for each non-terminal transition, and 0 for the terminal state (state 5).**
- **Discount Factor (γ): 1 (undiscounted problem).**
- **Initial Policy (π): Random policy where the agent chooses each action with equal probability (0.25 for each action).**
- **Terminal State: State 5 (after which the process terminates).**

```
s1  | s2  | s3
---------------
s4  | s5  | s6
---------------
s7  | s8  | s9
```

Initial Value Function (V_0):

At the beginning of the evaluation process, all state values are initialized to 0.

0	**0**	**0**
0	**0**	**0**
0	**0**	**0**

Step 2: Iteration 1 (k = 0)

Now, we calculate the value for each state using the Bellman Expectation Equation. The Bellman Expectation equation is given as:

$$V_1(s) = 0.25 \times [(-1 + \gamma \cdot V_0(s')) + (-1 + \gamma \cdot V_0(s')) + (-1 + \gamma \cdot V_0(s')) + (-1 + \gamma \cdot V_0(s'))]$$

Where:

- $\gamma = 1$ (undiscounted),
- $\pi(a|s) = 0.25$ (since the policy is random),
- $V_0(s') = 0$ for all neighboring states initially.

For state s5 (Terminal State):

Since state s5 is terminal, its value is 0 and it remains unchanged.

For state s4:

- Up: leads to s1, $V_0(s_1)=0$
- Down: leads to s7, $V_0(s_7)=0$
- Left: leads to outside the grid to s4, $V_0(s_4)=0$
- Right: leads to s5, $V_0(s_5)=0$

Value update for state $s4$:

$$V_1(4) = 0.25 \times [(-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0)]$$

$$V_1(4) = 0.25 \times (-4) = -1$$

For state s6:

- Up: leads to s3, V0(s3)=0
- Down: leads to s9, V0(s9)=0
- Left: leads to s5, V0(s5)=0
- Right: leads to outside the grid (we'll treat this as leading to a "ghost" state with value 0).

Value update for state s6:

$$V_1(6) = 0.25 \times [(-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0)]$$

$$V_1(6) = 0.25 \times (-4) = -1$$

For state s1:

- Up: Outside the grid (ghost state with value 0),
- Down: leads to s4, V0(s4)=0,
- Left: Outside the grid (ghost state with value 0),
- Right: leads to s2, V0(s2)=0.

Value update for state s1:

$$V_1(1) = 0.25 \times [(-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0) + (-1 + 1 \times 0)]$$

$$V_1(1) = 0.25 \times (-4) = -1$$

Value Function After Iteration 1 (k = 0)

-1	**-1**	**-1**
-1	**0**	**-1**
-1	**-1**	**-1**

Step 3: Iteration 2 (k = 1)

We now use the values from iteration 1 to calculate the values for iteration 2.

For state s4 (from Iteration 1, V1(s4)=−1):

- Up: leads to s1, V1(s1)=−1
- Down: leads to s7, V1(s7)=−1
- Left: leads to s3, V1(s3)=−1
- Right: leads to s5, V1(s5)=0

Value update for state s4:

$$V_2(4) = 0.25 \times [(-1 + 1 \times -1) + (-1 + 1 \times -1) + (-1 + 1 \times -1) + (-1 + 1 \times 0)]$$

$$V_2(4) = 0.25 \times (-1 - 2 - 2 - 1) = 0.25 \times (-6) = -1.5$$

For state $s6$ (from Iteration 1, $V_1(s6) = -1$):

- Up: leads to $s3$, $V_1(s3) = -1$
- Down: leads to $s9$, $V_1(s9) = -1$
- Left: leads to $s5$, $V_1(s5) = 0$
- Right: leads to outside the grid (ghost state with value 0).

Value update for state $s6$:

$$V_2(6) = 0.25 \times [(-1 + 1 \times -1) + (-1 + 1 \times -1) + (-1 + 1 \times 0) + (-1 + 1 \times 0)]$$

$$V_2(6) = 0.25 \times (-1 - 2 - 1 - 1) = 0.25 \times (-5) = -1.25$$

Value Function After Iteration 2 (k = 1)

-1	**-1**	**-1**
-1.5	**0**	**-1.25**
-1	**-1**	**-1**

b) Explain value iteration with an algorithm. **CO2 L3 7M**

Value Iteration is a dynamic programming algorithm used to compute the optimal value function V∗(s) and the corresponding optimal policy π∗ for a Markov Decision Process (MDP). It combines policy evaluation and policy improvement in a single step using the Bellman optimality equation.

Unlike policy iteration, value iteration skips full policy evaluation and instead uses truncated updates to estimate values, updating the value function directly using a max over actions.

3 Value iteration updates the value function using the **Bellman optimality equation**.

It stops when the value function changes very little between iterations (based on a small threshold θ).

It is more efficient than policy iteration in large or complex MDPs because it avoids full policy evaluation.

The algorithm guarantees convergence to the optimal value function V∗ and the corresponding policy π∗\pi^*π∗.

Works for **finite, discounted MDPs**.

Steps of the Value Iteration Algorithm

1. **Initialize**:
    1. Set V(s)=0 or any arbitrary values for all states s.
    2. Choose a small threshold θ>0 for stopping.
2. **Repeat Until Convergence**:

a)For each state s∈S:

1. Update the value function:

$$V(s) \leftarrow \max_a \sum_{s',r} P(s', r|s, a) \left[r + \gamma V(s')\right]$$

b)Stop when:

$$\max_s \left|V_{\text{new}}(s) - V_{\text{old}}(s)\right| < \theta$$

3.Extract Optimal Policy:

- **For each state s, compute:**

$$\pi^*(s) = \arg\max_a \sum_{s'\ r} P(s', r|s, a) \left[r + \gamma V(s')\right]$$

UNIT-III

6	**a)**	**Differentiate between SARSA and Q-learning.**	**CO3**	**L4**	**7M**

Feature	SARSA	Q-Learning
Policy Type	On-policy (follows current policy)	Off-policy (learns from optimal policy)
Update Target	Q(st+1,at+1)Q(s_{t+1}, a_{t+1})Q(st+1,at+1) (next move made)	max⁡aQ(st+1,a)\max_a Q(s_{t+1}, a)maxaQ(st+1,a) (best possible)
Behavior	Cautious (safe under exploration)	Optimistic (assumes best outcome)
Learning Style	Learns what we actually **do**	Learns what we **should do** optimally

b) Explain the Monte Carlo prediction with an example **CO3 L3 7M**

- Monte Carlo method is used to predict the state value 'v(s)' or the action value function 'Q(s,a)' from sampled episodes in the environment, where the model is unknown.
- It is a non-policy method, that learns from actual experience by averaging returns.

Types of Monte Carlo Prediction:

1. First-Visit Monte Carlo
   - Estimates $v_\pi(s)$ by averaging returns only after the first visit to state s in each episode.
2. Every-Visit Monte Carlo
   - Estimates $v_\pi(s)$ by averaging returns after every visit to state s in each episode.

Monte Carlo Prediction Algorithm (First-Visit):

Inputs:

- A policy π
- A set of episodes (sequences of states, actions, and rewards)

Steps:

1. Initialize:
   1. V(s) arbitrarily for all s∈S
   2. Returns(s) = [ ], for all s∈S
2. Loop:
   1. Generate an episode: $S_0,A_0,R_1,...,S_{T-1}A_{T-1}R_t$ following policy π.
   2. For each state s in the episode:
      1. Let G be the return following first visit to s
      2. Append G to Returns(s)

Update V(s) as the average of Returns(s) $$V(s) = \frac{1}{n}\sum_{t=1}^{n} G_t$$

Every-Visit Monte Carlo Algorithm:
Inputs:

- ❖ Policy π
- ❖ Episodes generated by following policy π.

Algorithm:
Initialize:

- ❖ V(s) arbitrarily for all s∈S
- ❖ Returns(s) = empty list for all states s∈S.

2. Loop:
   - ❖ Generate an episode: $S_0,A_0,R_1,...,S_{T-1}A_{T-1}R_t$.
   - ❖ For each state s in the episode:
     - Let G be the return (sum of discounted rewards) after that occurrence
     - Append G to Returns(s)

Update V(s)=average of Returns(s)

(OR)

7	**a)**	**Discuss the advantages of TD learning methods**	**CO3**	**L2**	**7M**

1. **Model-free:** TD does not require a model of the environment, making it flexible and applicable in many real-world situations.
2. **Low variance:** Since updates happen step-by-step based on immediate experience, TD tends to have lower variance compared to Monte Carlo methods, which rely on full episodes.
3. **Bootstrapping:** TD uses bootstrapping to update estimates after each step, leading to faster learning.
4. **Faster Convergence**: TD generally converges faster than MC because it updates its estimates incrementally after every step rather than waiting for a full episode.
5. **Online Learning:** TD can update values online, without waiting for the final outcome or terminal state.
6. **Handling Non-Episodic Tasks:** TD can be applied to non-episodic (continuing) tasks, where the environment doesn't have a defined terminal state.
7. **Less Memory and Computation**: Because it doesn't need to store the full episode, TD methods use less memory and computation**.**

	b)	**Derive the equation for the combination of MC error and TD error**	**CO3**	**L4**	**7M**

Step 1: MC Error definition

$$\text{MC Error} = G_t - V(S_t)$$

Step 2: Expand return G_t(recursive definition)

We know:

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

Substitute into MC error:

$$\text{MC Error} = (R_{t+1} + \gamma G_{t+1}) - V(S_t)$$

Step 3: Add and subtract $\gamma V(S_{t+1})$

(This is the trick to decompose it)

$$= R_{t+1} + \gamma G_{t+1} - V(S_t)$$

Add and subtract $\gamma V(S_{t+1})$:

$$= (R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) + \gamma(G_{t+1} - V(S_{t+1}))$$

Step 4: Identify terms

First term:

$$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

This is the **TD Error at time t**

Second term:

$$G_{t+1} - V(S_{t+1})$$

This is **MC Error at time t+1**

Step 5: Final Result

$$\boxed{\text{MC Error}_t = \text{TD Error}_t + \gamma\ \text{MC Error}_{t+1}}$$

Recall the key relation

$$\text{MC Error}_t = \text{TD Error}_t + \gamma\ \text{MC Error}_{t+1}$$

Step 1: Expand one more step

Substitute for MC Error_{t+1}:

$$\text{MC Error}_{t+1} = \text{TD Error}_{t+1} + \gamma\ \text{MC Error}_{t+2}$$

Now plug back:

$$\text{MC Error}_t = \text{TD Error}_t + \gamma(\text{TD Error}_{t+1} + \gamma\text{MC Error}_{t+2})$$

Step 2: Simplify

$$= \text{TD Error}_t + \gamma\text{TD Error}_{t+1} + \gamma^2\text{MC Error}_{t+2}$$

This matches your **Step 7 (Expansion of errors)**

Step 3: Keep expanding (pattern)

If we continue:

$$= \text{TD Error}_t + \gamma \text{TD Error}_{t+1} + \gamma^2 \text{TD Error}_{t+2} + \gamma^3 \text{MC Error}_{t+3}$$

Step 4: General form (Final Sum)

After expanding up to terminal time T:

$$\text{MC Error}_t = \sum_{k=0}^{T-t-1} \gamma^k \ \text{TD Error}_{t+k} + \gamma^{T-t} \ \text{MC Error}_T$$

This is exactly your **Step 8**

Step 5: Terminal condition

At terminal state T:

$$\text{MC Error}_T = 0$$

So the formula becomes:

$$\boxed{\text{MC Error}_t = \sum_{k=0}^{T-t-1} \gamma^k \ \text{TD Error}_{t+k}}$$

Unit-IV

8 a) Explain the backward view of the TD(λ) Algorithm with an example problem. **CO4 L3 7M**

TD(λ) Backward View Algorithm (Using Eligibility Traces)

Initialization:

- Initialize value function $V(s)$ arbitrarily for all states $s \in S$
  (but set $V(s) = 0$ if s is terminal)
- For each episode:
  - Initialize eligibility traces: $E(s) = 0$, for all $s \in S$
  - Initialize starting state S

Loop (for each step of the episode):

1. Choose action A according to policy π at state S
2. Take action A, observe:
   - reward R
   - next state S'
3. Compute TD error:

$$\delta \leftarrow R + \gamma V(S') - V(S)$$

4. Update eligibility trace:

$$E(S) \leftarrow E(S) + 1$$

5. For all states $s \in S$:
   - Update value function:
   $$V(s) \leftarrow V(s) + \alpha\delta E(s)$$
   - Decay eligibility trace:
   $$E(s) \leftarrow \gamma\lambda E(s)$$
6. Update current state:
   $$S \leftarrow S'$$
7. Repeat until S is terminal.

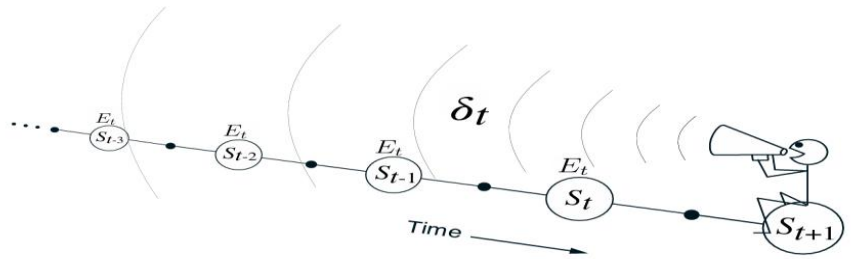


Figure 7.8: The backward or mechanistic view. Each update depends on the current TD error combined with eligibility traces of past events.

Example Execution:

Episode:

```
CSS

A → B → C → Terminal
```

Given:

- Rewards:
  - $R_1 = 0$
  - $R_2 = 0$
  - $R_3 = 1$
- Parameters:
  - $\alpha = 0.1$
  - $\lambda = 0.5$
  - $\gamma = 1$
- Initial values:
  - $V(A) = V(B) = V(C) = 0$
  - $E(s) = 0$ for all s

Step 1: A → B

- $R = 0, S = A, S' = B$
- $\delta = 0 + 1 \cdot V(B) - V(A) = 0$
- $E(A) = 0 + 1 = 1$
- Updates:
  - $V(A) = V(A) + 0.1 \cdot 0 \cdot 1 = 0$
  - Eligibility traces:
    - $E(A) = 1 \cdot 0.5 = 0.5$

Step 2: B → C

- $R = 0, S = B, S' = C$
- $\delta = 0 + 1 \cdot V(C) - V(B) = 0$
- $E(B) = 0 + 1 = 1$
- Updates:
  - $V(B) = V(B) + 0.1 \cdot 0 \cdot 1 = 0$
  - Eligibility traces:
    - $E(A) = 0.5 \cdot 0.5 = 0.25$
    - $E(B) = 1 \cdot 0.5 = 0.5$

Step 3: C → Terminal

- $R = 1, S = C, S' = Terminal$
- $\delta = 1 + 0 - V(C) = 1$
- $E(C) = 0 + 1 = 1$
- Updates:
  - $V(A) = 0 + 0.1 \cdot 1 \cdot 0.25 = 0.025$
  - $V(B) = 0 + 0.1 \cdot 1 \cdot 0.5 = 0.05$
  - $V(C) = 0 + 0.1 \cdot 1 \cdot 1 = 0.1$

Final Values After Episode:

- $V(A) = 0.025$
- $V(B) = 0.05$
- $V(C) = 0.1$

b) Explain the n-step TD prediction Algorithm with an example. Estimate the value function V(s) for a given policy using n-step TD. **CO4 L4 7M**

States: A → B → C → Terminal

Rewards: $R_1 = 0, R_2 = 0, R_3 = 1$

$\gamma = 1, \alpha = 0.1, n = 2$

Initial Value Estimates:

$$V(A) = 0, \quad V(B) = 0, \quad V(C) = 0$$

🔸 Episode Trajectory:

```
vbnet

Step 0: S₀ = A
Step 1: S₁ = B, R₁ = 0
Step 2: S₂ = C, R₂ = 0
Step 3: Terminal, R₃ = 1
```

🔸 n-Step Return Formula:

For n = 2, the return $G_t^{(2)}$ is:

$$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

✅ **At t = 0 (S_0 = A):**

- $R_1 = 0,\ R_2 = 0,\ S_2 = C,\ V(C) = 0$

$$G_0^{(2)} = 0 + 1 \cdot 0 + 1^2 \cdot V(C) = 0$$

$$V(A) = V(A) + \alpha \cdot (G_0^{(2)} - V(A)) = 0 + 0.1 \cdot (0 - 0) = 0$$

✅ **At t = 1 (S_1 = B):**

- $R_2 = 0,\ R_3 = 1,\ S_3 = Terminal,\ V(S_3) = 0$

$$G_1^{(2)} = 0 + 1 \cdot 1 + 1^2 \cdot V(\text{Terminal}) = 1$$

$$V(B) = V(B) + \alpha \cdot (G_1^{(2)} - V(B)) = 0 + 0.1 \cdot (1 - 0) = 0.1$$

✅ **At t = 2 (S_2 = C):**

Only 1 reward available (R_3 = 1), can't take full 2-step return, but we can treat it as a 1-step update:

$$G_2^{(1)} = R_3 + \gamma V(\text{Terminal}) = 1 + 0 = 1$$

$$V(C) = V(C) + 0.1 \cdot (1 - 0) = 0.1$$

Final Value Estimates After One Episode:

$$V(A) = 0$$

$$V(B) = 0.1$$

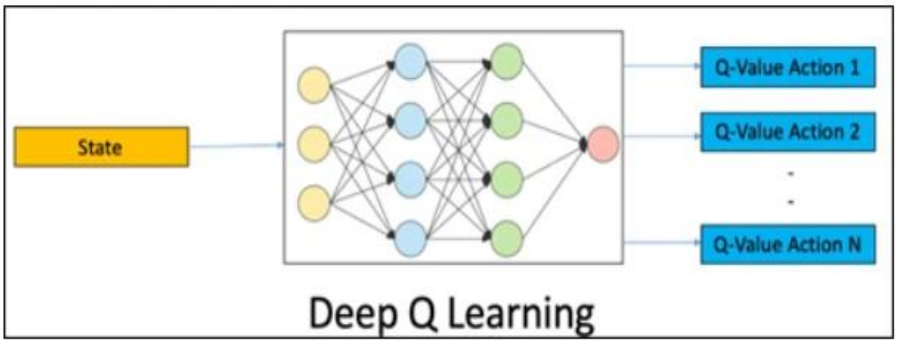
$$V(C) = 0.1$$

(OR)

9 a) Explain the architecture of DQN with a diagram **CO4 L2 7M**

Architecture of Deep Q Networks:

- **Input Layer:** A vector of numerical values representing the state information the input layer receives from the environment.
- **Hidden Layers:** Several fully connected neurons make up the DQN's hidden layers, which turn the input data into higher-level properties better suited for prediction.
- **Output Layer:** A single neuron in the DQN's output layer represents each potential action in the current state. These neurons' output values correspond to the estimated value of each action in that state.
- **Memory:** DQN employs a replay memory to retain the agent's training-related events. The present state, the action is taken, the reward received, and the subsequent state are all tuples that are stored in the memory.
- **Loss Function:** The DQN's loss function calculates how much the actual Q-values acquired from the replay memory differ from the predicted Q-values.
- **Optimization:** Update the network's weights to reduce the loss function as part of the DQN's optimization process. Usually, a form of stochastic gradient descent is used for this (SGD).



b) Explain the forward view of the TD(λ) Algorithm with an example problem. **CO4 L3 7M**

- It combines TD learning with Eligibility Traces (ET), borrowing ideas from both Monte Carlo (MC) and TD learning.
- It uses λ-returns, which are a weighted average of n-step returns, to update value estimates.
- The forward view is mainly theoretical. It looks ahead to future rewards to compute update targets, helping us understand how learning proceeds over multiple steps.

Averaging n-steps:

- The idea is to **combine multiple n-step returns into a single backup target**.

Example:
A backup could be a **weighted average** of a 2-step and a 4-step return:

$$\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$$

- **Weights must be positive and sum to 1** to guarantee convergence.

❖ The forward view **requires future rewards**, so it's not practical for online use but a **useful concept**.

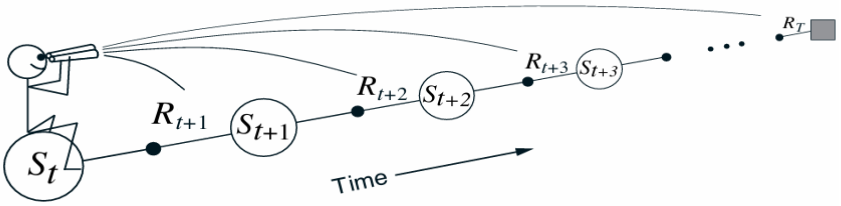


Figure 7.5: The forward or theoretical view. We decide how to update each state by looking forward to future rewards and states.

Problems with Forward View:

1. **Requires future knowledge**:
    1. To compute Gtλ, we must wait until the end of the episode (like MC).
2. **Computationally expensive**:
    1. Summing over all possible n-step returns is inefficient.

⟹ **Solution**: Use the **Backward View** (more practical).

Algorithm (Forward View TD(λ)):

Input: policy π, λ ∈ [0,1], step-size α

Initialize: V(s) arbitrary for all states s

For each episode:

1. Generate episode:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$$

2. For each time step $t = 0, 1, \ldots, T-1$:
    - Compute all n-step returns $G_{t:t+n}$ for $n = 1$ to $T - t$
    - Compute λ-return:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t} V(S_T)$$

    - Update value function:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t^\lambda - V(S_t) \right]$$

Episode: A → B → C → Terminal
Rewards:

- **R1=1 R2=2**

Initial values:

❖ **V(A)=0, V(B)=0, V(C)=0 α=0.1, λ=0.5, γ=1**

Step-by-step calculation for state A:

1-step return from A:

$$G_{t:t+1} = R_1 + \gamma V(B) = 1 + 1 \cdot 0 = 1$$

2-step return:

$$G_{t:t+2} = R_1 + \gamma R_2 + \gamma^2 V(C) = 1 + 1 \cdot 2 + 1^2 \cdot 0 = 3$$

λ-return:

$$G_t^\lambda = (1-\lambda)G_{t:t+1} + \lambda G_{t:t+2} = (1-0.5)(1) + 0.5(3) = 0.5 \cdot 1 + 0.5 \cdot 3 = 0.5 + 1.5 = 2.0$$

Value update:

$$V(A) = V(A) + \alpha[G_t^\lambda - V(A)] = 0 + 0.1(2.0 - 0) = 0.1 \cdot 2.0 = 0.2$$

Prepared By:
G.V. LEELA KUMARI
Asst. Professor
Dept Of AIML

Signature of the Faculty

1.

2.