

DL – Unit 6 (Reinforcement Learning) – END-SEM PYQ Answers

MAY-JUNE 2023

Q7a Explain Markov Decision Process with Markov property.

[6 Marks]

Markov Decision Process (MDP)

A Markov Decision Process is the formal mathematical framework used to model sequential decision-making problems in reinforcement learning. It provides a way to describe environments where an agent takes actions, receives rewards, and transitions between states, with the goal of learning an optimal policy.

Formal Definition — The 5-Tuple (S, A, P, R, γ)

- **S — State Space:** The complete set of all possible situations (states) the agent can be in. Example: in a chess game, S is the set of all board configurations.
- **A — Action Space:** The complete set of actions available to the agent. Example: in chess, A is all possible legal moves.
- **P — Transition Probability:** $P(s' | s, a)$ = probability of moving to state s' when taking action a in state s . This defines the environment's dynamics.
- **R — Reward Function:** $R(s, a, s')$ = the immediate scalar reward received after transitioning from state s to s' via action a . The agent's objective is to maximise cumulative reward.
- **γ — Discount Factor:** $\gamma \in [0, 1]$ discounts future rewards. $\gamma=0$ means the agent is myopic (cares only about immediate reward). $\gamma \rightarrow 1$ means the agent values future rewards almost as much as immediate ones.

The Markov Property

The Markov property states that the future state of the system depends only on the current state and action, not on the entire history of past states and actions. Formally:

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) \\ = P(s_{t+1} | s_t, a_t)$$

'The future is independent of the past given the present.'

This means the current state s_t encodes all relevant information from history needed to make optimal decisions.

MDP Interaction Loop



Key Concepts

- **Policy (π):** A mapping from states to actions — $\pi(a|s) = P(\text{take action } a \text{ in state } s)$. The agent's behaviour strategy.
- **Return (G_t):** The discounted cumulative reward from time t onward: $G_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \dots$
- **Value Function $V^\pi(s)$:** Expected return starting from state s and following policy π : $V^\pi(s) = E_\pi[G_t | s_t = s]$.
- **Q-Function $Q^\pi(s,a)$:** Expected return starting from s , taking action a , then following π : $Q^\pi(s,a) = E_\pi[G_t | s_t = s, a_t = a]$. Also called the action-value function.

Note: The MDP framework requires the Markov property to hold. In practice, many real-world problems are Partially Observable MDPs (POMDPs) where the agent does not directly observe the full state s — instead, it receives an observation o that is a (possibly noisy) function of s . Deep RL handles this by using the history of observations or a learned internal state (via RNNs) as the effective state.

Q7b Explain in detail Dynamic programming algorithms for reinforcement learning. **[6 Marks]**

Dynamic Programming (DP) Algorithms for RL

Dynamic Programming refers to a set of algorithms that compute optimal policies for MDPs by exploiting the recursive structure of the Bellman equations. DP requires complete knowledge of the environment's dynamics (transition probabilities P and reward function R) — which is often unavailable in practice but provides the theoretical foundation for all RL methods.

The Bellman Equations

Bellman Expectation Equation (for policy π):

$$V^\pi(s) = \sum_a \pi(a|s) \cdot \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V^\pi(s')]$$

Bellman Optimality Equation (for optimal policy π^*):

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V^*(s')] \quad Q^*(s,a) = \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

Algorithm 1: Policy Evaluation

Policy Evaluation iteratively computes the value function V^π for a fixed policy π by repeatedly applying the Bellman expectation equation until convergence.

Policy Evaluation (Iterative):

Initialise $V(s) = 0$ for all s

Repeat until $|V_{\text{new}} - V_{\text{old}}| < \epsilon$ (convergence threshold):

For each state s :

$$V_{\text{new}}(s) = \sum_a \pi(a|s) \cdot \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V(s')] \quad V \leftarrow V_{\text{new}}$$

Algorithm 2: Policy Improvement

Given a value function V^π , a better policy π' can be derived by acting greedily with respect to V^π — always choosing the action that maximises the expected return.

Policy Improvement:

For each state s :

$$\pi'(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V^\pi(s')]$$

Policy Improvement Theorem guarantees: $V^{\pi'}(s) \geq V^\pi(s)$ for all s .

Algorithm 3: Policy Iteration

Policy Iteration alternates between Policy Evaluation and Policy Improvement until the policy no longer changes — at that point, the policy is guaranteed to be optimal.

Policy Iteration:

Initialise π arbitrarily

Repeat:

1. Policy Evaluation: compute V^π
2. Policy Improvement: $\pi' \leftarrow \text{greedy}(V^\pi)$
3. If $\pi' = \pi \rightarrow \text{STOP}$ (optimal policy found) Else: $\pi \leftarrow \pi'$

Algorithm 4: Value Iteration

Value Iteration combines the evaluation and improvement steps into a single update — applying the Bellman optimality operator directly. It converges faster than policy iteration in practice.

Value Iteration:

Initialise $V(s) = 0$ for all s

Repeat until convergence:

For each state s :

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

Extract optimal policy: $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a) \cdot [R(s,a,s') + \gamma V(s')]$

| Property | Policy Iteration vs Value Iteration |
|--------------------|---|
| Convergence | Policy Iteration: fewer iterations (polynomial in $ S A $). Value Iteration: more iterations but each is cheaper. |
| Per-iteration Cost | Policy Iteration: full policy evaluation is expensive. Value Iteration: single Bellman update per iteration. |
| When to Use | Policy Iteration: when state space is small. Value Iteration: when approximate answers are acceptable early. |

Note: DP algorithms require the full model (P and R) and become intractable for large state spaces (curse of dimensionality). Model-free RL methods (Q-Learning, SARSA, Policy Gradient) address this by learning from sampled experience rather than computing over all states.

Q7c Explain Simple reinforcement learning for Tic-Tac-Toe.

[5 Marks]

Reinforcement Learning Applied to Tic-Tac-Toe

Tic-Tac-Toe is a classic simple environment used to illustrate RL concepts because it has a small, finite state space that can be enumerated exactly. The RL formulation maps naturally onto the MDP framework.

MDP Formulation for Tic-Tac-Toe

- **State (s):** The configuration of the 3×3 board — which cells are X, O, or empty. There are at most 5,478 unique legal board states.
- **Action (a):** Placing an X (or O) in any empty cell. The action space shrinks as the game progresses.
- **Reward (R):** Sparse reward — +1 for winning, -1 for losing, 0 for a draw or non-terminal move.
- **Policy (π):** Determines which cell to mark given the current board state.

Learning Algorithm — Value-based RL

Training procedure (self-play or against random opponent):

1. Initialise $V(s) = 0.5$ for all non-terminal states

$V(\text{win}) = 1.0, V(\text{loss}) = 0.0, V(\text{draw}) = 0.5$

2. For each game:
 - a. Observe current state s
 - b. With prob ϵ : take random action (exploration)
With prob $1-\epsilon$: take action leading to s' with max $V(s')$ (exploitation)
 - c. After taking action and observing next state s' :
 $V(s) \leftarrow V(s) + \alpha \cdot [V(s') - V(s)]$ (Temporal Difference update)
 - d. If game over, receive terminal reward and update $V(\text{final_state})$

α = learning rate (e.g., 0.1) ϵ = exploration rate (decreases over time)

Key Observations

- **Convergence:** After enough games (self-play or against a fixed opponent), $V(s)$ converges to the true win probability from each state under the optimal policy.
- **Exploration vs Exploitation:** The ϵ -greedy strategy balances exploring new moves (to discover better strategies) with exploiting currently known good moves.
- **Temporal Difference Learning:** The update rule $V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$ propagates reward information backwards through the game sequence without waiting for the final outcome — this is the core TD learning idea, also used in Q-Learning.

Note: This simple RL agent for Tic-Tac-Toe was demonstrated in Sutton & Barto's seminal textbook 'Reinforcement Learning: An Introduction'. It serves as a microcosm of all RL concepts: state representation, action selection, credit assignment, and policy improvement through experience.

Q8a Write Short Note on Q Learning and Deep Q-Networks.

[6 Marks]

Q-Learning

Q-Learning (Watkins, 1989) is a model-free, off-policy reinforcement learning algorithm that directly learns the optimal action-value function $Q^*(s, a)$ without requiring knowledge of the environment's transition dynamics.

Q-Learning Update Rule

Q-Learning Update (Temporal Difference):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

The bracketed term is the TD Error (δ):

$$\delta = r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)$$

= Target - Current estimate

α = learning rate (how fast to update)

γ = discount factor (how much to value future rewards)

Action selection: ϵ -greedy — random with prob ϵ , else $\arg\max_a Q(s, a)$

- **Tabular Q-Learning:** Maintains an explicit Q-table of size $|S| \times |A|$. Works perfectly for small discrete environments (Tic-Tac-Toe, FrozenLake) but is completely infeasible for large or continuous state spaces (e.g., Atari games with raw pixel input have $\sim 10^6$ dimensional state spaces).

Deep Q-Networks (DQN)

Deep Q-Network (Mnih et al., DeepMind 2015) extends Q-Learning to complex environments by using a deep neural network (convolutional + dense layers) to approximate the Q-function: $Q(s, a; \theta) \approx Q^*(s, a)$, where θ are the network weights.

DQN Architecture (for Atari games)

Input: 4 stacked grayscale frames (84×84×4) ← captures motion

→ Conv(32, 8×8, stride 4) + ReLU

→ Conv(64, 4×4, stride 2) + ReLU

→ Conv(64, 3×3, stride 1) + ReLU

→ Flatten → Dense(512) + ReLU

→ Dense(|A|) ← one output per action (e.g., 18 for Atari)

Output: Q-value estimate for each possible action

DQN Key Innovations

- **Experience Replay:** Instead of updating on consecutive transitions (which are highly correlated and cause unstable training), DQN stores all (s, a, r, s') transitions in a replay buffer of size ~1 million. During training, random mini-batches are sampled — breaking temporal correlations and improving data efficiency.
- **Target Network:** A second 'frozen' copy of the network θ^- provides stable Q-targets. The target Q-value is computed as: $y = r + \gamma \cdot \max_{a'} Q(s', a'; \theta^-)$. The target network θ^- is updated by copying θ every C steps (e.g., C=10,000). Without this, both the target and predictions change simultaneously, causing instability.
- **Loss Function:** Mean Squared Error between predicted Q-values and target Q-values: $L(\theta) = E[(y - Q(s,a;\theta))^2]$. Huber loss is often used instead of MSE to reduce sensitivity to outliers.

DQN Extensions

- **Double DQN:** Decouples action selection from action evaluation to reduce overestimation bias. Uses the online network to select the best action and the target network to evaluate it.
- **Dueling DQN:** Separates Q into V(s) (state value) + A(s,a) (advantage of each action). Better at estimating state values when many actions have similar effects.
- **Prioritized Experience Replay:** Samples transitions with high TD error more frequently, improving sample efficiency.

Note: DQN achieved superhuman performance on 49 Atari games using only raw pixel input and game score as reward — the first deep learning system to do so. It demonstrated that deep neural networks can serve as effective function approximators in RL, opening the door to Deep Reinforcement Learning as a field.

Q8b What are the challenges of reinforcement learning? Explain any four in detail. **[6 Marks]**

Challenges of Reinforcement Learning

- **1. Sample Inefficiency:** RL agents require an enormous number of environment interactions to learn effective policies. DQN needs ~50 million frames (the equivalent of ~38 days of play time) to achieve human-level Atari performance. This makes RL extremely expensive for real-world applications where data collection involves physical hardware, safety constraints, or significant time. Solutions include model-based RL (learning an internal environment model to simulate experience), transfer learning, and imitation learning.
- **2. Reward Specification and Reward Hacking:** Designing a reward function that truly captures the desired behaviour is extremely difficult. Agents often find unintended ways to

maximise the reward without achieving the intended goal (reward hacking). Example: a boat racing agent learned to collect bonus rings in circles rather than finishing the race. Reward misspecification also poses alignment risks — an agent optimising the wrong reward can behave in unexpected and dangerous ways.

- **3. Exploration vs. Exploitation Dilemma:** The agent must balance exploring new states/actions to discover better strategies with exploiting what it currently knows to maximise reward. Too much exploitation leads to getting stuck in locally good but globally suboptimal policies. Too much exploration wastes time on uninformative actions. In sparse reward environments (e.g., the reward only arrives at the end of a long sequence), random exploration almost never discovers the reward signal — requiring sophisticated exploration strategies like curiosity-driven exploration, intrinsic motivation, or count-based exploration.
- **4. Credit Assignment Problem:** When a sequence of many actions leads to a final reward, the agent must determine which of those actions deserved credit for the outcome. In a chess game spanning 50 moves, only the final checkmate generates a reward — how should the agent attribute credit to the move 30 moves earlier that set up the position? Temporal Difference learning addresses this partially, but long credit assignment chains remain challenging.
- **5. Non-stationarity:** In multi-agent settings, or when the agent's own improving policy changes the effective environment dynamics, the learning problem becomes non-stationary — the optimal Q-values shift as the agent improves, requiring continuous adaptation.
- **6. Generalisation:** RL agents often learn policies that are brittle to changes in the environment — even minor visual changes (different textures, lighting) can cause performance collapse. Building policies that generalise to unseen situations remains an open problem.

Note: The combination of sample inefficiency and reward specification challenges is why RL has had limited real-world deployment compared to supervised learning. Sim-to-real transfer (training in simulation, deploying on real hardware) is a key practical approach that mitigates data collection costs but introduces the 'reality gap' as a new challenge.

Q8c What is deep reinforcement learning? Explain in detail.

[5 Marks]

Deep Reinforcement Learning (Deep RL)

Deep Reinforcement Learning is the combination of deep neural networks with reinforcement learning algorithms. The deep network serves as a function approximator, enabling RL to scale to problems with high-dimensional state and action spaces (such as raw pixel images, continuous control, or natural language) that are completely intractable for tabular RL methods.

Why Deep Networks in RL?

- **High-dimensional states:** A raw Atari game frame is $84 \times 84 \times 3 \approx 21,168$ dimensions. Storing a Q-table over all possible frames is impossible. A CNN can map this to a compact feature representation.
- **Continuous action spaces:** In robotics, the action is a continuous vector (joint torques). Deep networks can parametrize policies over continuous spaces directly.
- **End-to-end learning:** Deep RL can learn directly from raw sensory data (pixels, audio, proprioception) to actions, without hand-engineered features.

Categories of Deep RL Algorithms

- **Value-Based Methods:** Approximate $Q^*(s,a)$ with a neural network. Example: DQN, Double DQN, Dueling DQN, Rainbow DQN. Work well for discrete action spaces.

- **Policy Gradient Methods:** Directly parametrize the policy $\pi(a|s; \theta)$ as a neural network and optimise it by gradient ascent on the expected return. Example: REINFORCE, Actor-Critic (A2C), PPO (Proximal Policy Optimization), TRPO. Work for both discrete and continuous actions.
- **Actor-Critic Methods:** Combine value-based and policy gradient — the Actor (policy network) selects actions, the Critic (value network) evaluates them. Reduces variance of policy gradient estimates. Example: A3C (Asynchronous Advantage Actor-Critic), SAC (Soft Actor-Critic), TD3.
- **Model-Based Deep RL:** Learn a neural network model of environment dynamics $P(s'|s,a)$ and use it for planning or generating synthetic experience. Example: Dreamer (latent-space imagination), AlphaZero (MCTS + neural network).

Landmark Deep RL Results

| System | Achievement |
|----------------------|--|
| DQN (DeepMind, 2015) | Superhuman Atari performance from raw pixels. |
| AlphaGo (2016) | Defeated world champion Go player — previously thought decades away. |
| AlphaZero (2017) | Mastered Chess, Shogi, Go from self-play alone in 24 hours. |
| OpenAI Five (2019) | Defeated world champions at Dota 2 — complex multi-agent game. |
| AlphaStar (2019) | Grandmaster-level StarCraft II play. |
| ChatGPT RLHF (2022) | Used RL from Human Feedback to align language model behaviour. |

Note: RLHF (Reinforcement Learning from Human Feedback) is now one of the most important applications of Deep RL — it is the technique used to align large language models like ChatGPT, Claude, and Gemini to human preferences, using human comparison data to train a reward model, then fine-tuning the LLM with PPO to maximise that reward.

NOV-DEC 2023

Q7a Explain dynamic programming algorithms for reinforcement learning. **[6 Marks]**

[REPEATED] Dynamic Programming algorithms (Policy Evaluation, Policy Improvement, Policy Iteration, Value Iteration) are covered in Q7b of May-June 2023 with Bellman equations and pseudocode for all four algorithms. Refer to that section.

Q7b What is deep reinforcement learning? Explain in detail. **[6 Marks]**

[REPEATED] Deep Reinforcement Learning is covered comprehensively in Q8c of May-June 2023, including the motivation, categories (Value-based, Policy Gradient, Actor-Critic, Model-based), and landmark results (DQN, AlphaGo, RLHF).

Q7c Explain Simple reinforcement learning for Tic-Tac-Toe. **[5 Marks]**

[REPEATED] RL for Tic-Tac-Toe is covered in Q7c of May-June 2023, including MDP formulation (state, action, reward) and the value-based TD learning algorithm with exploration via ϵ -greedy.

Q8a Explain Markov decision process. **[6 Marks]**

[REPEATED] Markov Decision Process is covered in Q7a of May-June 2023 with the full 5-tuple definition, Markov property, agent-environment interaction loop diagram, and key concepts (policy, return, value function, Q-function).

Q8b Write Short Note on Q Learning and Deep Q-Networks. **[6 Marks]**

[REPEATED] Q-Learning and Deep Q-Networks (DQN) are covered in Q8a of May-June 2023 with the update rule, replay buffer, target network, DQN architecture, and extensions (Double DQN, Dueling DQN, Prioritized Replay).

Q8c What are the challenges of reinforcement learning? Explain any four in detail. **[5 Marks]**

[REPEATED] Challenges of Reinforcement Learning (sample inefficiency, reward specification/hacking, exploration-exploitation, credit assignment, non-stationarity, generalisation) are covered in Q8b of May-June 2023 with detailed explanations.

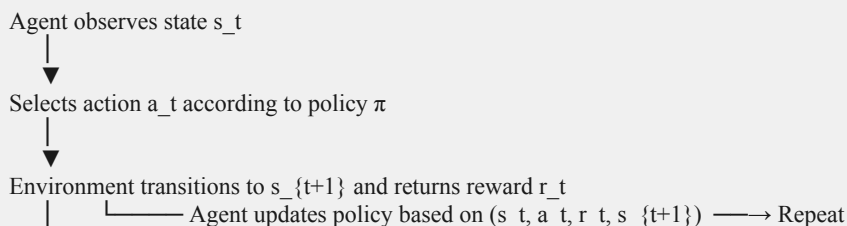
MAY-JUNE 2024

Q7a What is Reinforcement Learning? State and explain its advantages and disadvantages. **[6 Marks]**

Reinforcement Learning — Definition

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make sequential decisions by interacting with an environment. Unlike supervised learning (which requires labelled examples) or unsupervised learning (which seeks structure in data), RL learns from feedback in the form of a reward signal — receiving positive rewards for desirable actions and negative rewards (penalties) for undesirable ones. The agent's goal is to learn a policy (a mapping from states to actions) that maximises the expected cumulative discounted reward over time.

Core Loop



Advantages of Reinforcement Learning

- **No labelled data required:** RL learns from its own experience through interaction — no expensive human annotation of training examples is needed. The reward signal provides implicit supervision.
- **Handles sequential decisions:** RL naturally models problems where actions have long-term consequences — chess, robotics, resource management — which are beyond the scope of supervised learning.

- **Achieves superhuman performance:** RL agents (AlphaGo, AlphaZero, OpenAI Five) have surpassed human experts in games once considered exclusively human intellectual achievements.
- **Continual learning:** RL agents can continue to improve by gathering new experience, adapting to changes in the environment over time.
- **Discovers novel strategies:** RL often discovers unexpected, creative solutions that human experts would not have found (e.g., AlphaZero's non-human chess openings).

Disadvantages of Reinforcement Learning

- **Sample inefficiency:** Requires millions of interactions to learn what humans learn in minutes. Computationally expensive for real-world deployment.
- **Reward design difficulty:** Designing a correct reward function is a non-trivial engineering challenge. Poor reward design leads to reward hacking or unsafe behaviour.
- **Training instability:** Deep RL training is notoriously brittle — small changes in hyperparameters or environment can cause catastrophic performance drops.
- **Lack of interpretability:** Neural network policies are black boxes — it is hard to understand why an agent took a specific action, which is critical for safety-critical applications.
- **No transfer between tasks:** RL agents typically learn task-specific policies that do not transfer well to new environments, requiring retraining from scratch.
- **Exploration challenges:** In sparse-reward environments, naive exploration methods almost never stumble upon the reward, making learning extremely slow.

Note: The best practical results with RL combine it with human knowledge — through reward shaping, imitation learning (initialise from human demonstrations), and curriculum learning (start with easy versions of the task and progressively increase difficulty). RLHF (used in ChatGPT, Claude) is a successful example of combining human feedback with RL to achieve aligned, useful behaviour.

Q7b What are different types of Reinforcement Learning? Explain in brief.

[6 Marks]

Types of Reinforcement Learning

- **Model-Free RL:** The agent learns directly from interaction with the environment without building an explicit model of the environment's transition dynamics. This is the most common approach. Examples: Q-Learning, SARSA, DQN, PPO, SAC.
- **Model-Based RL:** The agent learns or is given a model of the environment ($P(s'|s,a)$ and $R(s,a)$) and uses it for planning — simulating future states without real interaction. More sample efficient but requires accurate models. Examples: Dyna-Q, AlphaZero (uses MCTS + learned model), World Models, Dreamer.
- **Value-Based RL:** Learns an explicit value function (Q or V) and derives the policy implicitly by acting greedily. Best for discrete action spaces. Examples: Q-Learning, DQN, Double DQN.
- **Policy-Based (Policy Gradient) RL:** Directly parametrizes and optimises the policy without maintaining a value function. Handles continuous action spaces naturally. Examples: REINFORCE, PPO, TRPO.
- **Actor-Critic RL:** Combines both — an Actor (policy) generates actions, a Critic (value function) evaluates them and provides a lower-variance learning signal. Examples: A3C, SAC, TD3, A2C.
- **On-Policy RL:** Learns about and updates the policy being used to generate experience. More stable but less sample efficient. Examples: SARSA, PPO, A3C.

- **Off-Policy RL:** Learns about a different policy from the one being used to generate experience (can reuse old data via replay buffer). More sample efficient. Examples: Q-Learning, DQN, SAC, TD3.
- **Multi-Agent RL (MARL):** Multiple agents interact in a shared environment, which may be cooperative (AlphaStar units), competitive (GAN training), or mixed. Each agent's optimal strategy depends on the others — a game-theoretic setting.

Note: The on-policy vs off-policy distinction is fundamental in practice: off-policy methods (DQN, SAC) can reuse experience from a replay buffer, making them much more sample efficient. On-policy methods (PPO) require fresh experience at each update but are often more stable and easier to tune for new environments.

Q7c Compare Active and Passive Reinforcement Learning.

[5 Marks]

Active vs Passive Reinforcement Learning

This distinction refers to whether the agent has control over the actions it takes during learning.

| Property | Passive vs Active Reinforcement Learning |
|---------------|---|
| Definition | Passive RL: The agent follows a fixed policy π and learns (evaluates) how good that policy is — it has no control over actions. Active RL: The agent must choose actions itself and improve its policy through exploration. |
| Agent Control | Passive: No control — the policy is externally provided, agent just observes outcomes. Active: Full control — the agent selects actions at each step. |
| Goal | Passive: Policy Evaluation — compute $V^\pi(s)$ or $Q^\pi(s,a)$ for the given π . Active: Policy Optimisation — find the best policy π^* that maximises return. |
| Exploration | Passive: No exploration dilemma — the fixed policy determines all actions. Active: Must balance exploration (trying new actions to discover better ones) vs exploitation (using the best known action). |
| Algorithms | Passive: Temporal Difference evaluation (TD(0), TD(λ)), Monte Carlo evaluation — these compute value estimates for a fixed policy. Active: Q-Learning, SARSA, Policy Gradient, DQN — these actively improve the policy. |
| Analogy | Passive: A student who watches a teacher solve problems and learns to evaluate their approach. Active: A student who must solve problems themselves and learn from trial-and-error feedback. |

Note: Passive RL corresponds to the 'Prediction Problem' in RL theory — given a policy, predict the value function. Active RL corresponds to the 'Control Problem' — find the optimal policy. In practice, most real-world RL is active. Passive RL techniques are used as subroutines within actor-critic methods (the Critic performs passive policy evaluation for the current actor's policy).

Q8a Write short note on Deep Q-Learning. [6 Marks]

[REPEATED] Deep Q-Learning (DQN) is covered in Q8a of May-June 2023 with the Q-Learning update rule, DQN architecture for Atari, Experience Replay, Target Network, and extensions (Double DQN, Dueling DQN, Prioritized Replay).

Q8b What are different characteristics of Reinforcement Learning? [6 Marks]

Characteristics of Reinforcement Learning

- **Trial-and-Error Learning:** The agent is not told the correct action to take. It must discover good actions by trying them and observing the resulting reward — a fundamentally different learning signal from supervised labels.
- **Delayed Reward (Temporal Credit Assignment):** Actions may have long-term consequences. The reward received at a given moment may be the result of an action taken many steps earlier. The agent must learn to assign credit appropriately across time.
- **Sequential Decision Making:** RL deals with a sequence of interdependent decisions, where each action changes the state and thus the options available for future actions. Current choices constrain future possibilities.
- **Goal-Directed Behaviour:** The agent has an explicit objective — maximise the cumulative discounted reward $G_t = \sum \gamma^k \cdot r_{t+k}$. All learning is driven by this single objective.
- **Agent-Environment Interaction:** RL is defined by the interaction loop between the agent (the decision-maker) and the environment (everything the agent affects and is affected by). The boundary between agent and environment is a key modelling decision.
- **Exploration-Exploitation Trade-off:** The agent must explore the state space to discover better strategies while simultaneously exploiting current knowledge to earn rewards. This tension is fundamental to all RL problems.
- **Online Learning:** RL agents typically learn incrementally as they gather new experience (online), rather than being trained on a fixed pre-collected dataset like in supervised learning.
- **Non-Stationarity:** The agent itself changes over time (as it learns), which changes its behaviour, which in turn changes the distribution of states it visits — making the learning problem inherently non-stationary.

Note: These characteristics collectively define what makes RL both powerful and challenging. The combination of sequential decisions, delayed rewards, and no direct supervision makes RL harder to train than supervised learning, but also applicable to a wider range of real-world problems — anywhere an agent must learn from interaction rather than from pre-labeled examples.

Q8c Explain in detail Dynamic programming algorithms for reinforcement learning. **[5 Marks]**

[REPEATED] Dynamic Programming algorithms for RL are covered in Q7b of May-June 2023, including the Bellman equations and pseudocode for all four algorithms: Policy Evaluation, Policy Improvement, Policy Iteration, and Value Iteration.
