UNIT–5 : Reinforcement Learning and Genetic Algorithms (PYQ Solution)

AKTU · Machine Learning Techniques (BCDS062) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. What is a Markov Decision Process (MDP)? (May 2026)

An MDP is the mathematical model of reinforcement learning, defined by a tuple (S, A, P, R, γ): states, actions, transition probabilities P(s′ | s, a), rewards R(s, a) and discount factor γ. The Markov property says the next state depends only on the current state and action, not on history. The goal is a policy π that maximises expected discounted return.

Q2. What is Q-learning? (Jan 2026)

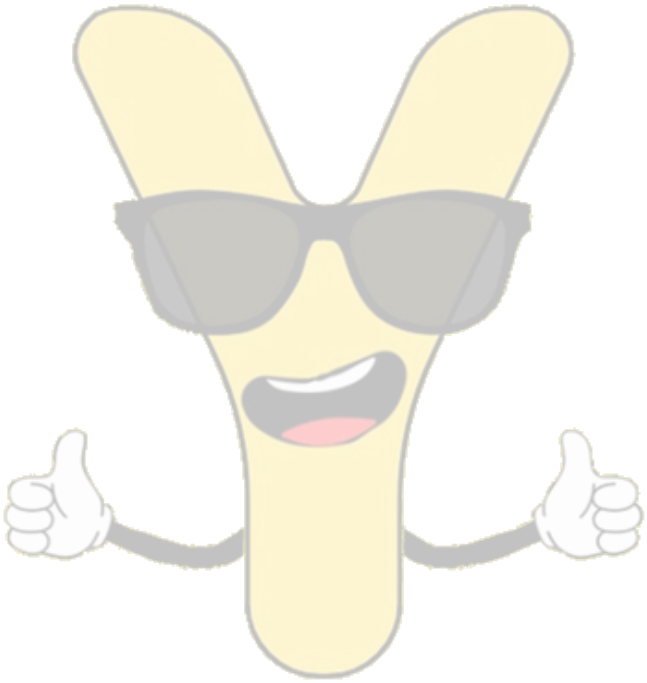
Q-learning is a model-free, off-policy reinforcement learning algorithm that learns the action-value function Q(s, a), the expected return of taking action a in state s and acting optimally afterwards. It updates by Q(s, a) ← Q(s, a) + α [r + γ max Q(s′, a′) − Q(s, a)], without needing a model of the environment.

Q3. Describe the role of the reward function in reinforcement learning. (Jan 2025)

The reward function gives the agent a numeric signal r after each action, telling it how good the outcome was. It defines the goal: the agent learns a policy that maximises the cumulative (discounted) reward. It guides exploration and learning, so a badly designed reward can lead to unwanted behaviour. Example: +1 for winning a game, −1 for losing.

Q4. Define Offspring, Chromosome and Gene in a Genetic Algorithm. (Jan 2024)

A chromosome is one candidate solution encoded as a string (e.g. binary 10110). A gene is one element or position of the chromosome (a single bit or parameter). Offspring are new chromosomes produced from parents by crossover and mutation, forming the next generation.



SECTION B Answers (≈250 Words Each)

Q1. Describe Q-Learning, including the Q-function and the steps of the algorithm. What is the role of Q-learning in learning optimal actions and policies? (May 2026, Jan 2025)

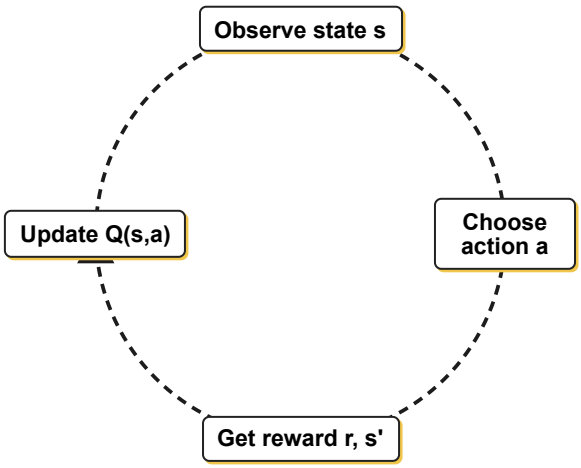
Q-learning is a model-free reinforcement learning method in which the agent learns the quality of actions without knowing the environment's transition or reward model.

Q-function: Q(s, a) is the expected cumulative discounted reward when taking action a in state s and following the optimal policy afterwards. It relates to the value function: V*(s) = max over a of Q(s, a). The optimal policy is π*(s) = argmax over a of Q(s, a), so once Q is learned, the agent simply picks the action with the highest Q-value.

Update rule (Bellman):
Q(s, a) ← Q(s, a) + α [r + γ max over a′ of Q(s′, a′) − Q(s, a)]
α = learning rate, γ = discount factor (0 to 1), r = immediate reward, s′ = next state. The bracket is the **temporal-difference error**.

Algorithm:

1. Initialise the Q-table Q(s, a) to zeros (or small random values).
2. Observe the current state s.
3. Choose action a using **ε-greedy**: random with probability ε (exploration), otherwise argmax Q(s, a) (exploitation).
4. Perform a; receive reward r and next state s′.
5. Update Q(s, a) with the rule above.
6. Set s ← s′; repeat until the episode ends, then run many episodes.



Q-learning loop

Numerical illustration: Q(s, a) = 0, r = 5, max Q(s′, ·) = 10, α = 0.5, γ = 0.9 → Q = 0 + 0.5 (5 + 9 − 0) = **7**.

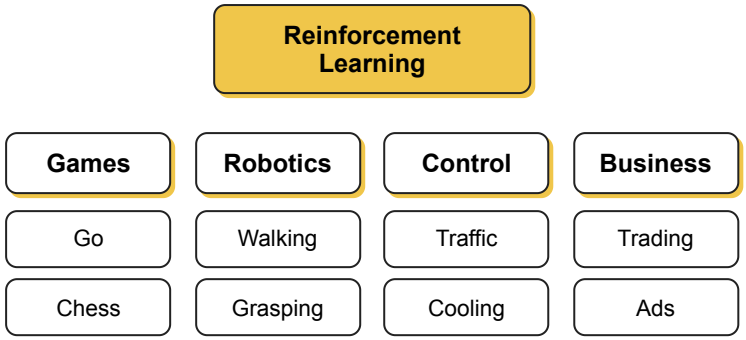
Role: because Q-learning is off-policy and converges to Q* if every state-action pair is visited often, it lets the agent learn optimal actions purely from trial and error, balancing exploration and exploitation.

Q2. Discuss the applications of reinforcement learning. In which problems is it used? (Jan 2024)

Reinforcement learning (RL) suits problems where an agent makes a **sequence of decisions**, gets delayed reward and no labelled answers exist.

Applications:

- **Games:** AlphaGo, chess, Atari, backgammon (TD-Gammon).
- **Robotics:** walking, grasping, navigation and manipulation learned by trial and error.
- **Autonomous vehicles:** lane keeping, parking, path planning.
- **Resource management:** data-centre cooling, job scheduling, traffic signal control, network routing.
- **Finance:** portfolio management and algorithmic trading.
- **Recommendation and advertising:** personalised content, bidding.
- **Healthcare:** treatment planning and dosage policies.
- **Natural language:** dialogue systems and fine-tuning language models with human feedback.
- **Industry:** supply chain and inventory control.



RL applications

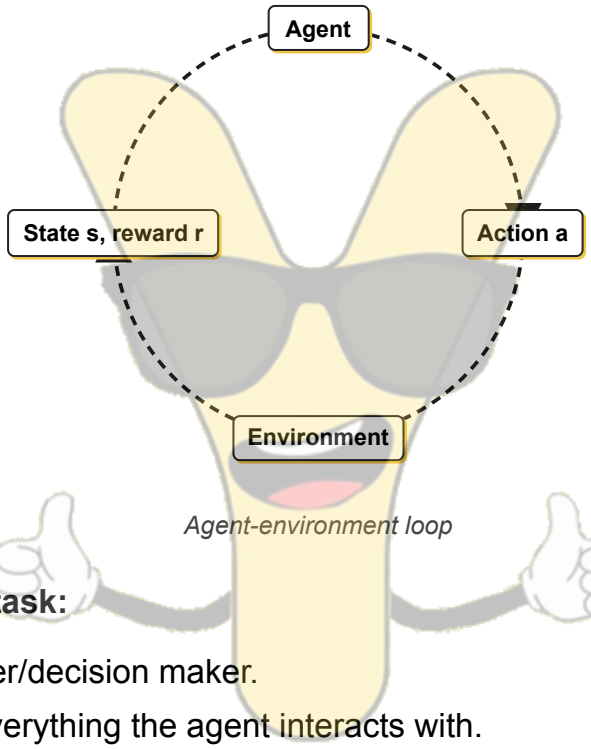
Suitable when: the environment can be simulated, feedback is a scalar reward, and there is a trade-off between exploring and exploiting. It is not suitable when plentiful labelled data exists (use supervised learning) or when errors are very costly during exploration.



SECTION C Answers (≈500 Words Each)

Q1. Explain reinforcement learning with the structure of an RL task and a Markov Decision Process. How does it differ from supervised and unsupervised learning? (May 2026, Jan 2026, Jan 2025)

Reinforcement learning (RL) is learning by interaction: an **agent** observes the **state** of an **environment**, takes an **action**, and receives a **reward** and the next state. The agent aims to learn a **policy** that maximises the total (discounted) reward over time.



Agent-environment loop

Structure of an RL task:

- **Agent:** the learner/decision maker.
- **Environment:** everything the agent interacts with.
- **State (s)**, **action (a)**, **reward (r)**, **policy π(s) → a**, **value function V(s)**, **Q(s, a)**, and optionally a model of the environment.
- **Return:** G = r1 + γ r2 + γ² r3 + … with discount γ.

Markov Decision Process (MDP): the formal model (S, A, P, R, γ). P(s′ | s, a) is the transition probability and the **Markov property** holds: the future depends only on the present state. The optimal value satisfies the **Bellman equation**: V*(s) = max over a of [R(s, a) + γ Σ P(s′ | s, a) V*(s′)]. Solving it by value iteration, policy iteration or Q-learning gives the optimal policy.

Examples: a robot reaching a goal in a grid (reward +10 at goal, −1 per step, −10 for collision); chess (reward at win/loss); a thermostat learning to save energy.

Basis	Supervised	Unsupervised	Reinforcement
Feedback	Correct labels	None	Reward (delayed, evaluative)
Data	Fixed labelled set	Fixed unlabelled set	Generated by agent's actions
Goal	Predict	Find structure	Maximise cumulative reward
Special issue	Overfitting	Evaluation	Exploration vs exploitation

Conclusion: RL suits sequential decision problems where the agent must learn from consequences of its own actions.

Q2. Describe the Q-Learning algorithm, its key terms, features, elements and applications. (Jan 2026, Jan 2024)

Q-learning is a model-free, off-policy temporal-difference algorithm that learns the optimal action-value function Q*(s, a).

Key terms:

- **Agent, environment, state, action, reward.**
- **Q-value** Q(s, a): expected future reward for action a in state s.
- **Q-table:** a table of Q-values, one entry per state-action pair.
- **Learning rate α** (0 to 1), **discount factor γ** (0 to 1), **ε** for ε-greedy exploration.
- **Policy:** π(s) = argmax over a of Q(s, a).

Key features: model-free (no transition model needed); off-policy (learns the greedy policy while behaving differently); converges to Q* if all pairs are visited infinitely often; handles stochastic environments; deep Q-networks replace the table for large state spaces.

Update rule: Q(s, a) ← Q(s, a) + α [r + γ max over a′ of Q(s′, a′) − Q(s, a)]

Algorithm:

1. Initialise Q(s, a) arbitrarily (zeros).
2. For each episode: start in state s.
3. Choose a using ε-greedy; execute it, observe r and s′.
4. Update Q(s, a) by the rule.
5. s ← s′; stop at a terminal state; repeat for many episodes.

Example: a grid with a goal giving r = 100, γ = 0.9, α = 1. A cell next to the goal gets Q = 100; the one before it gets 0 + 0.9 × 100 = 90; then 81, and so on, so Q-values

decrease with distance from the goal and the agent follows increasing values.

Applications: game playing (Atari via DQN), robot path planning, traffic light control, elevator scheduling, inventory management, recommender systems and autonomous driving.

Conclusion: Q-learning learns optimal behaviour from experience alone.

Q3. Explain the Genetic Algorithm: definition, components, working with a flowchart and GA cycle. Discuss the roles of chromosomes, genes and population. (Jan 2026, Jan 2025, Jan 2024)

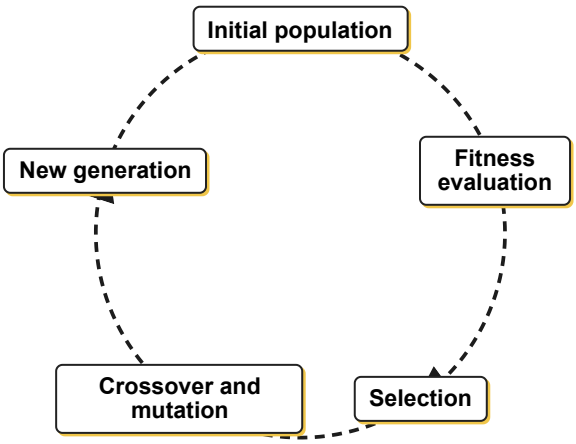
A **Genetic Algorithm (GA)** is a search and optimisation technique inspired by natural selection: a population of candidate solutions evolves over generations using selection, crossover and mutation.

Components:

- **Chromosome:** one encoded candidate solution (e.g. binary string 10110).
- **Gene:** one position/parameter in the chromosome.
- **Population:** the set of chromosomes in a generation.
- **Fitness function:** measures solution quality.
- **Selection:** fitter chromosomes get more chance to become parents (roulette wheel, tournament).
- **Crossover:** swaps parts of two parents to create offspring.
- **Mutation:** randomly flips genes to keep diversity.

GA cycle:

1. Initialise a random population.
2. Evaluate the fitness of each chromosome.
3. Select parents.
4. Apply crossover (probability about 0.7-0.9) and mutation (about 0.01) to create offspring.
5. Form the new generation (replace the population).
6. Repeat from step 2 until the stopping condition (max generations or satisfactory fitness).



GA cycle

Example: maximise f(x) = x² for x in 0..31 with a 5-bit encoding. Population: 01101 (13, f = 169), 11000 (24, f = 576), 01000 (8, f = 64), 10011 (19, f = 361). Total = 1170, so 11000 has selection probability 49%. Crossover of 11000 and 10011 after bit 2 gives 11011 (27, f = 729) and 10000 (16, f = 256). Fitness of the best member rises across generations toward 11111 (31, f = 961).

Conclusion: GA finds good solutions without gradients, suited to complex, non-linear search spaces.

Q4. What are crossover and mutation in Genetic Algorithms, and how are they applied in Genetic Programming and real-world problems? (May 2026)

Crossover (recombination) combines the genetic material of two parents to produce offspring that inherit features of both. **Mutation** randomly alters genes in an offspring, preserving diversity and preventing premature convergence.

Crossover types:

- **Single-point:** parents 11001|011 and 00110|110 swap tails → 11001110 and 00110011.
- **Two-point:** swaps the segment between two cut points.
- **Uniform:** each gene is taken from either parent with equal probability.

Mutation types: bit flip (10110 → 10010), swap, inversion, or Gaussian change for real-valued genes. Rate is kept low (about 1 / chromosome length).



Producing offspring

Genetic Programming (GP): chromosomes are **program trees** (functions as internal nodes, terminals as leaves), e.g. (x + 3) × y. **Crossover** swaps randomly chosen subtrees between two parent programs. **Mutation** replaces a random subtree with a new random subtree or changes a node's function. Fitness is how well the program solves the task on test cases. GP evolves programs for symbolic regression, controllers and classifiers.

Real-world applications:

- Scheduling and timetabling, travelling salesperson routing.
- Neural network weight and structure optimisation.
- Engineering design (antenna, aircraft wing shapes).
- Feature selection, portfolio optimisation, game strategy and robotics.

Conclusion: crossover exploits good building blocks and mutation explores new ones; together they drive evolutionary search in both GA and GP.

