Reinforcement Learning: A Comprehensive Question Bank

Based on "A Comprehensive Analysis of Core Reinforcement Learning Algorithms"

July 12, 2025

Contents

Part I

Multiple-Choice Questions (4 Options)

1 The Foundations of Reinforcement Learning

1. In the agent-environment framework, what does the agent receive from the environment at each time step t?

   (a) A policy π_t and a value function V_t.

   (b) **A state s_t and a reward r_t.**

   (c) The transition probability function P.

   (d) The discount factor γ.

2. What is the primary goal of a reinforcement learning agent?

   (a) To learn the environment's model.

   (b) To visit as many states as possible.

   (c) To maximize the immediate reward.

   (d) **To maximize the cumulative reward over the long run.**

3. Which of the following defines a finite Markov Decision Process (MDP)?

   (a) (S, A, P, γ)

   (b) (S, A, R, π)

   (c) (S, A, P, R, γ)

   (d) (S, A, V, Q, π)

4. What does the Markov Property state?

   (a) The future is dependent on the entire history of states and actions.

   (b) The reward depends only on the current state.

   (c) **The future is independent of the past, given the present state.**

   (d) The policy must be deterministic.

5. What is the key difference between the immediate reward R_t and the return G_t?

   (a) The reward is a scalar, while the return is a vector.

(b) The reward is for continuous tasks, and the return is for episodic tasks.

(c) **The reward is immediate feedback, while the return is the long-term cumulative reward.**

(d) The reward is always positive, while the return can be negative.

6. What is the purpose of the discount factor γ?

   (a) To make the agent prioritize immediate rewards over future rewards.

   (b) To ensure the return is always a finite sum in episodic tasks.

   (c) **To determine the present value of future rewards and ensure convergence in continuous tasks.**

   (d) To control the exploration-exploitation trade-off.

7. A discount factor γ close to 1 makes the agent:

   (a) **Farsighted, placing significant weight on future rewards.**

   (b) Myopic, prioritizing immediate rewards.

   (c) Behave randomly.

   (d) Learn the model more quickly.

8. What is a policy, π?

   (a) The agent's memory of past states.

   (b) The true value of a state.

   (c) **A mapping from states to actions, defining the agent's behavior.**

   (d) The probability of transitioning to a new state.

9. What does the state-value function, $V^{\pi}(s)$, quantify?

   (a) The immediate reward for being in state s.

   (b) The probability of reaching state s.

   (c) **The expected cumulative return starting from state s and following policy π.**

   (d) The best possible action to take from state s.

10. What is the primary advantage of learning the action-value function, $Q^*(s,a)$, over the state-value function, $V^*(s)$?

    (a) $Q^*(s,a)$ is easier to compute.

    (b) $V^*(s)$ cannot be learned in a model-free way.

(c) $Q^*(s,a)$ **allows for model-free control by choosing the action with the highest value.**

(d) $Q^*(s,a)$ does not require a discount factor.

11. The identity $V^*(s) = \max_a Q^*(s,a)$ means:

    (a) The optimal policy is to choose the action with the lowest Q-value.

    (b) **The value of a state under an optimal policy is the value of the best action from that state.**

    (c) The V-function and Q-function are always equal.

    (d) This identity only holds for deterministic policies.

12. What is the set of all possible states denoted by?

    (a) A

    (b) P

    (c) S

    (d) R

13. The state transition probability $P(s'|s,a)$ defines:

    (a) The agent's policy.

    (b) The reward function.

    (c) **The dynamics, or "model," of the environment.**

    (d) The value of a state.

14. A stochastic policy $\pi(a|s)$ outputs:

    (a) A single action for a given state.

    (b) **A probability distribution over actions for a given state.**

    (c) The expected return for a given state.

    (d) The next state.

15. The return G_t for an episodic task is defined as:

    (a) R_{t+1}

    (b) γR_{t+1}

    (c) $R_{t+1} + R_{t+2} + \cdots + R_T$

    (d) $\max_a R(s,a)$

2 Model-Based Planning with Dynamic Programming

16. Dynamic Programming methods require what crucial piece of information?

    (a) A set of sample episodes.

    (b) An initial policy.

    (c) **A perfect model of the environment (P and R).**

    (d) A constant step-size parameter α.

17. The Bellman equations express a recursive relationship between:

    (a) The policy and the value function.

    (b) **The value of a state and the values of its successor states.**

    (c) The immediate reward and the discount factor.

    (d) The agent and the environment.

18. What is the key difference between the Bellman expectation equation and the Bellman optimality equation?

    (a) The expectation equation uses a discount factor, while the optimality equation does not.

    (b) **The optimality equation includes a $\max$ operator over actions, while the expectation equation averages over the policy.**

    (c) The expectation equation is for V-functions, and the optimality equation is for Q-functions.

    (d) The optimality equation is used in model-free methods, while the expectation equation is used in model-based methods.

19. The Value Iteration algorithm iteratively applies which update rule?

    (a) The Bellman expectation equation.

    (b) The policy improvement theorem.

    (c) **The Bellman optimality equation as an update rule.**

    (d) The incremental mean update.

20. In Value Iteration, what does the value function $V_k(s)$ represent?

    (a) The value of state s after k episodes.

    (b) **The optimal value achievable from state s with a horizon of k steps.**

    (c) The value of state s under a random policy for k iterations.

(d) The k-th best value for state s.

21. Why is Value Iteration guaranteed to converge to V^*?

    (a) Because the state space is finite.

    (b) Because the policy improves at each step.

    (c) **Because the Bellman operator is a contraction mapping for $\gamma < 1$.**

    (d) Because it uses a small threshold θ.

22. After Value Iteration converges to V^*, how is the optimal policy π^* extracted?

    (a) By running more iterations.

    (b) The policy is the last value function V_k.

    (c) **By performing a one-step lookahead and choosing the action that maximizes expected return using V^*.**

    (d) By using an ϵ-greedy strategy.

23. Policy Iteration alternates between which two steps?

    (a) Value update and policy extraction.

    (b) **Policy evaluation and policy improvement.**

    (c) Exploration and exploitation.

    (d) Model learning and planning.

24. The "policy evaluation" step in Policy Iteration aims to:

    (a) Find the optimal policy.

    (b) **Compute the state-value function V^π for the current policy π.**

    (c) Update the policy to be greedy.

    (d) Check if the policy has converged.

25. The "policy improvement" step in Policy Iteration involves:

    (a) Iteratively solving the Bellman expectation equation.

    (b) Averaging returns from sample episodes.

    (c) **Making the policy greedy with respect to the current value function V^π.**

    (d) Decreasing the value of ϵ.

26. What does the Policy Improvement Theorem guarantee?

(a) That Policy Iteration converges in one iteration.

(b) That the value function is always increasing.

(c) **That the new greedy policy is a strict improvement over the old one, unless it's already optimal.**

(d) That the model of the environment is accurate.

27. How does the computational cost per iteration of Policy Iteration (PI) generally compare to Value Iteration (VI)?

    (a) PI is always cheaper.

    (b) **PI is generally more expensive due to the iterative policy evaluation step.**

    (c) They have the same computational cost.

    (d) The cost depends on the discount factor γ.

28. In terms of the number of iterations to converge, how does Policy Iteration (PI) typically compare to Value Iteration (VI)?

    (a) PI typically requires many more iterations.

    (b) **PI often converges in far fewer iterations.**

    (c) They require the same number of iterations.

    (d) The number of iterations is unpredictable for both.

29. Generalized Policy Iteration (GPI) refers to the general process of:

    (a) Only performing policy evaluation.

    (b) **Letting policy evaluation and policy improvement processes interact to find an optimal solution.**

    (c) Using a general-purpose function approximator.

    (d) A specific implementation of Value Iteration.

30. In the grid world example, the values "ripple" outwards from the terminal states. This visualizes:

    (a) How the policy changes over time.

    (b) **How value information propagates through the state space with each VI iteration.**

    (c) The process of policy extraction.

    (d) The effect of a stochastic policy.

31. The Bellman optimality equation for V^* is a system of:
    (a) Linear equations.
    (b) **Non-linear equations.**
    (c) Differential equations.
    (d) Parametric equations.

32. The policy evaluation step in PI solves for V^π using the:
    (a) Bellman Optimality Equation.
    (b) **Bellman Expectation Equation.**
    (c) Incremental Mean Update rule.
    (d) Q-learning update rule.

33. Value Iteration can be seen as a special case of Policy Iteration where:
    (a) The policy is always random.
    (b) The discount factor is 1.
    (c) **The policy evaluation step is truncated to just one Bellman backup.**
    (d) The policy improvement step is skipped.

34. The final output of the Value Iteration algorithm is:
    (a) The optimal policy π^*.
    (b) **The optimal state-value function V^*.**
    (c) The optimal action-value function Q^*.
    (d) A set of sample trajectories.

3 Model-Free Learning from Experience

35. What is the primary characteristic of model-free learning algorithms?
    (a) They are less efficient than model-based methods.
    (b) They only work for deterministic environments.
    (c) **They do not require knowledge of the environment's dynamics (P and R).**
    (d) They do not use value functions.

36. Monte Carlo (MC) methods estimate value functions by:

(a) Solving the Bellman equations.

(b) **Averaging the returns from many complete sample episodes.**

(c) Performing one-step lookaheads.

(d) Learning a model and then planning.

37. A key limitation of standard Monte Carlo methods is that they are only defined for:

    (a) Continuous tasks.

    (b) **Episodic tasks.**

    (c) Deterministic policies.

    (d) Known models.

38. What does it mean that Monte Carlo methods do not "bootstrap"?

    (a) They cannot be run on a computer.

    (b) They require an initial value function to start.

    (c) **They update a state's value based on the full return, not on other estimated values.**

    (d) They are biased.

39. What is the difference between First-Visit MC and Every-Visit MC?

    (a) First-Visit is on-policy, Every-Visit is off-policy.

    (b) **First-Visit averages returns for the first visit to a state in an episode; Every-Visit averages returns for all visits.**

    (c) First-Visit is for V-functions, Every-Visit is for Q-functions.

    (d) First-Visit is biased, Every-Visit is unbiased.

40. Why must MC control methods learn the action-value function $Q(s, a)$ instead of $V(s)$?

    (a) $Q(s, a)$ has lower variance.

    (b) $V(s)$ cannot be learned from episodes.

    (c) **Improving the policy from $Q(s, a)$ is model-free, whereas improving from $V(s)$ requires a model.**

    (d) $Q(s, a)$ converges faster.

41. The incremental update rule $V \leftarrow V + \alpha(G - V)$ is used to:

(a) Perform policy improvement.

(b) **Efficiently update the mean return without storing all past returns.**

(c) Extract the optimal policy.

(d) Calculate the Bellman error.

42. Using a constant step-size α in the incremental update is particularly useful for:

(a) Episodic tasks only.

(b) **Non-stationary environments where dynamics might change.**

(c) Guaranteeing convergence to the true sample average.

(d) Reducing the bias of the estimate.

43. The exploration-exploitation dilemma refers to the trade-off between:

(a) Model-based and model-free methods.

(b) Bias and variance.

(c) **Maximizing reward with current knowledge vs. trying new actions to find better strategies.**

(d) On-policy and off-policy learning.

44. An ϵ-greedy policy is a common way to:

(a) Speed up convergence of Value Iteration.

(b) **Ensure continued exploration in model-free control.**

(c) Evaluate a given policy.

(d) Reduce the variance of Monte Carlo estimates.

45. What defines an on-policy learning algorithm?

(a) It uses two different policies for behavior and learning.

(b) It requires a model of the environment.

(c) **It evaluates and improves the same policy that is used to generate experience.**

(d) It can only learn from expert demonstrations.

46. What defines an off-policy learning algorithm?

(a) It is always less efficient than on-policy learning.

(b) **It evaluates a target policy that is different from the behavior policy used to collect data.**

(c) It does not suffer from the exploration-exploitation dilemma.

(d) It can only be used in episodic tasks.

47. Which of the following is a classic example of an on-policy algorithm?

    (a) Value Iteration.

    (b) **SARSA.**

    (c) Q-Learning.

    (d) Monte Carlo Prediction.

48. Which of the following is a classic example of an off-policy algorithm?

    (a) Policy Iteration.

    (b) First-Visit MC.

    (c) SARSA.

    (d) **Q-Learning.**

49. The "target" in the Monte Carlo update rule $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{Target} - Q(s,a))$ is:

    (a) The immediate reward R_{t+1}.

    (b) The value of the next state $V(S_{t+1})$.

    (c) **The complete return from the episode G_t.**

    (d) Zero.

50. Model-free methods are generally considered to have:

    (a) Low sample efficiency.

    (b) **High sample inefficiency (i.e., they require many samples).**

    (c) The same sample efficiency as model-based methods.

    (d) No need for samples.

51. The update rule $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$ belongs to which family of algorithms?

    (a) Dynamic Programming.

    (b) Monte Carlo.

    (c) **Temporal-Difference (TD) Learning.**

    (d) Policy Gradient methods.

52. How does Temporal-Difference (TD) learning combine aspects of DP and MC?

    (a) It requires a model and works on episodes.

(b) It is model-based and does not bootstrap.
(c) **It is model-free like MC and bootstraps like DP.**
(d) It has high bias and high variance.

53. The term $R_{t+1} + \gamma V(S_{t+1})$ is known as the:

    (a) Monte Carlo return.
    (b) Bellman expectation.
    (c) **TD Target.**
    (d) Policy improvement term.

54. Compared to Monte Carlo methods, TD learning typically has:

    (a) Higher variance.
    (b) **Lower variance.**
    (c) The same variance.
    (d) No variance.

4 Synthesis and Comparative Analysis

55. The main advantage of Dynamic Programming over Monte Carlo methods is:

    (a) Its ability to learn without a model.
    (b) Its low computational cost.
    (c) **Its high sample efficiency (as it doesn't use samples).**
    (d) Its applicability to continuous state spaces.

56. The main advantage of Monte Carlo methods over Dynamic Programming is:

    (a) **Their ability to learn without a model.**
    (b) Their low variance.
    (c) Their fast convergence speed in terms of iterations.
    (d) Their use of bootstrapping.

57. Bootstrapping in RL refers to:

    (a) Starting the algorithm with random values.
    (b) **Updating a value estimate based on other value estimates.**
    (c) Averaging returns from multiple episodes.
    (d) Using a soft policy for exploration.

58. Which algorithm family is characterized by high variance and low bias?

    (a) Dynamic Programming.

(b) **Monte Carlo Methods.**
(c) Temporal-Difference Learning.
(d) All of the above.

59. Which algorithm family is characterized by low variance (from sampling) and potential bias from bootstrapping?
    (a) **Dynamic Programming.**
    (b) Monte Carlo Methods.
    (c) Model-free methods in general.
    (d) Off-policy methods.

60. You would choose a Dynamic Programming approach if:
    (a) The environment's model is unknown.
    (b) The task is non-episodic.
    (c) **You have a perfect model and the state space is manageably small.**
    (d) You need to balance exploration and exploitation.

61. You would choose a Monte Carlo approach if:
    (a) **The environment's model is unknown and the task is episodic.**
    (b) You need the fastest possible convergence.
    (c) The state space is extremely large.
    (d) You want to avoid high variance.

62. The term "planning" in RL is synonymous with:
    (a) **Model-based methods.**
    (b) Model-free methods.
    (c) On-policy learning.
    (d) Off-policy learning.

63. The term "learning" in RL is most often associated with:
    (a) Dynamic Programming.
    (b) **Model-free methods.**
    (c) Using a perfect model.
    (d) Policy evaluation.

64. TD learning is often preferred over MC methods in practice because:
    (a) It is simpler to implement.
    (b) It is unbiased.
    (c) **It can learn online from incomplete episodes and has lower variance.**

(d) It does not require a discount factor.

65. The "curse of dimensionality" primarily affects which aspect of the algorithms discussed?

    (a) The discount factor.

    (b) **The feasibility of storing value tables and sweeping through states in DP.**

    (c) The exploration rate ϵ.

    (d) The calculation of a single return in MC.

66. If an agent learns a model from experience and then uses DP on that learned model, this is called:

    (a) Model-free learning.

    (b) **Model-based reinforcement learning (or indirect RL).**

    (c) Off-policy learning.

    (d) Temporal-Difference learning.

67. The update for which algorithm must wait until the end of an episode?

    (a) Value Iteration.

    (b) Policy Iteration.

    (c) Temporal-Difference Learning.

    (d) **Monte Carlo Methods.**

68. The update for which algorithm can happen after a single time step?

    (a) First-Visit Monte Carlo.

    (b) Every-Visit Monte Carlo.

    (c) **Temporal-Difference Learning.**

    (d) All model-free methods.

69. The core idea of acting greedily with respect to a value function is central to:

    (a) Policy evaluation.

    (b) **Policy improvement.**

    (c) The Markov property.

    (d) The definition of return.

70. In the grid world example, the value of a state far from the terminal states takes many iterations to converge because:

    (a) The discount factor is too low.

    (b) The policy is stochastic.

    (c) **Value Iteration propagates value information only one step at a time.**

(d) The rewards are too small.

71. Which two algorithms are guaranteed to converge to the optimal policy in a finite MDP?

    (a) Monte Carlo and TD Learning.
    (b) **Value Iteration and Policy Iteration.**
    (c) On-policy MC and Off-policy MC.
    (d) All of the above.

72. The "TD Error" is the difference between:

    (a) The current policy and the optimal policy.
    (b) The V-function and the Q-function.
    (c) **The TD Target and the current value estimate.**
    (d) The first-visit return and the every-visit return.

73. Which algorithm directly solves a system of non-linear equations for the optimal value function?

    (a) Policy Iteration.
    (b) Monte Carlo.
    (c) **Value Iteration (iteratively).**
    (d) SARSA.

74. Which algorithm's policy evaluation step involves solving a system of linear equations?

    (a) **Policy Iteration.**
    (b) Value Iteration.
    (c) Q-Learning.
    (d) Monte Carlo.

Part II

Explanatory Questions and Answers

5 Foundations and Dynamic Programming

1. **Question:** Explain the Markov Property and why it is crucial for simplifying the decision-making process in reinforcement learning.

   Answer:

   - **Definition:** The Markov Property states that the future is independent of the past, given the present. Mathematically, this means the probability of transitioning to the next state s_{t+1} depends only on the current state s_t and the current action a_t, not on the entire history of preceding states and actions. The state s_t is a sufficient statistic of the future.
   - **Importance:** This property is crucial because it immensely simplifies the agent's decision-making process. If the Markov property holds, the agent does not need to remember every event that has ever happened to it. It can select its optimal action by considering only its current state, as that state encapsulates all relevant information from the history. This avoids the "curse of history" and makes the problem computationally tractable, allowing us to use the MDP framework and its associated Bellman equations.

2. **Question:** Describe the two roles of the discount factor, γ, and how adjusting its value from near 0 to near 1 changes the agent's behavior.

   Answer:

   - **Role 1 (Mathematical Convenience):** For continuous (non-terminating) tasks, the sum of future rewards could diverge to infinity. The discount factor $\gamma \in [0, 1)$ ensures that the infinite sum of discounted rewards (the return) is a finite value, as long as the rewards are bounded. This makes the return a well-defined quantity that algorithms can optimize.
   - **Role 2 (Behavioral Influence):** The value of γ shapes the agent's strategy.
     - A γ value close to 0 makes the agent **"myopic"** or "short-sighted." Future rewards are heavily discounted, so the agent prioritizes maximizing its immediate reward.
     - A γ value close to 1 makes the agent **"farsighted."** It places significant weight on future rewards, enabling it to make long-term plans and potentially sacrifice immediate rewards for greater cumulative rewards later on.

3. **Question:** Compare and contrast Value Iteration and Policy Iteration, focusing on their computational cost per iteration, number of iterations to converge, and the core update being performed.

   Answer:

   - **Core Update:**
     - **Value Iteration (VI):** Performs a single, simple update per state in each iteration. It directly applies the Bellman *optimality* equation as an assignment statement, propagating value information one step at a time.
     - **Policy Iteration (PI):** Performs two complex steps. First, it does a full **policy evaluation**, which is an iterative process in itself that solves the Bellman *expectation* equation until the value function for the current policy converges. Second, it does a **policy improvement** step.
   - **Computational Cost per Iteration:**
     - **VI:** Each iteration is computationally cheap, involving a single sweep through the state space with complexity around $O(|A||S|^2)$.
     - **PI:** Each iteration is very expensive. The policy evaluation step can require many sweeps or solving a system of $|S|$ linear equations, which can have a complexity up to $O(|S|^3)$.
   - **Number of Iterations to Converge:**
     - **VI:** Often requires many iterations to converge, as values propagate slowly.
     - **PI:** Often converges in a surprisingly small number of iterations, as each policy improvement step makes a large, decisive jump to a provably better policy.

4. **Question:** What is the Bellman optimality equation for $Q^*(s, a)$? Deconstruct the equation and explain what each part represents.

   Answer: The Bellman optimality equation for $Q^*(s, a)$ is:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right)$$

   Deconstruction:

   - $Q^*(s, a)$**:** The left side is the value we want to find: the optimal value of taking action a in state s and then acting optimally thereafter.
   - $\sum_{s'} P(s'|s, a)[\ldots]$**:** This is an expectation. Since the environment is stochastic, we don't know for sure which state s' we will land in. This part averages over all possible next states, weighted by their transition probabilities $P(s'|s, a)$.
   - $R(s, a, s')$**:** This is the immediate reward received for transitioning from state s to state s' after taking action a.

- **$\gamma \max_{a'} Q^*(s', a')$:** This is the discounted value of the future. It says that once we land in the next state s', we will continue to act optimally. The $\max_{a'}$ operator chooses the best possible action from that next state, and $Q^*(s', a')$ gives its value. This future value is then discounted by γ.

In words, the equation states that the optimal value of a state-action pair is the expected immediate reward plus the expected discounted value of the best possible action from the next state.

5. **Question:** Explain the process of "policy extraction" after Value Iteration has converged. Why is this step necessary?

   Answer:

   - **Process:** After the Value Iteration algorithm has run for enough iterations, it produces a stable, optimal state-value function, V^*. However, V^* itself doesn't tell the agent what to do. Policy extraction is the process of using V^* to find the optimal policy, π^*. For each state s, one performs a one-step lookahead. For every possible action a, one calculates the expected return: $\sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma V^*(s'))$. The optimal action for state s is the one that maximizes this value.
   - **Necessity:** This step is necessary because Value Iteration only computes *how good* each state is, not *which action* to take. The algorithm's output is the optimal value function, not the optimal policy. The policy is implicit in the value function, and the extraction step makes it explicit. This requires the model (P and R) for the one-step lookahead.

6 Model-Free Learning

6. **Question:** What is the fundamental distinction between planning and learning in RL? Provide an example for each.

   Answer:

   - **Distinction:** The fundamental distinction is the availability of the environment's model ($P(s'|s, a)$ and $R(s, a, s')$).
     - **Planning (Model-Based):** The agent has access to a perfect model. It uses this model to compute an optimal policy without necessarily interacting with the real environment.
     - **Learning (Model-Free):** The agent does not have a model. It must learn an optimal policy by directly interacting with the environment to collect experience (samples of states, actions, and rewards).
   - **Examples:**

    - **Planning:** A chess program like Deep Blue. The rules of chess are perfectly known (this is the model). The program can use algorithms like Value Iteration to plan by simulating move sequences internally.
    - **Learning:** An agent learning to play a new Atari video game for the first time. The agent doesn't know the rules or physics of the game. It must learn by pressing buttons (actions) and observing the change on the screen (next state) and the score (reward).

7. **Question:** Explain what "bootstrapping" is. Which algorithm families use it, which do not, and what are the consequences of using it?

   Answer:

   - **Definition:** Bootstrapping, in the context of RL, means updating a value estimate for a state based on the existing estimates of other states.
   - **Who Uses It?**
     - **Dynamic Programming (VI, PI):** Yes. The update for $V_k(s)$ is based on the values from the previous iteration, $V_{k-1}(s')$.
     - **Temporal-Difference (TD) Learning:** Yes. The update for $V(S_t)$ is based on the current estimate of the next state's value, $V(S_{t+1})$.
     - **Monte Carlo (MC) Methods:** No. The update for $V(S_t)$ is based on the actual, complete return G_t from an episode, which is independent of any other value estimate.
   - **Consequences:** The main consequence is a trade-off. Bootstrapping introduces **bias** into the estimates, because the updates are based on other estimates which may be inaccurate. However, it dramatically reduces **variance** and increases learning speed, as the agent doesn't have to wait for the final outcome of a long episode. It allows for online learning from incomplete sequences.

8. **Question:** Describe the exploration-exploitation dilemma. How does an ϵ-greedy policy attempt to solve it?

   Answer:

   - **Dilemma:** The exploration-exploitation dilemma is a fundamental challenge in control problems. The agent must choose between two competing goals:
     - **Exploitation:** Use its current knowledge to choose the action it believes is best, thereby maximizing its immediate reward.
     - **Exploration:** Choose a different, seemingly suboptimal action to gather more information about the environment, with the hope of discovering an even better long-term strategy.

     A purely greedy agent might get stuck in a local optimum, never discovering the true best policy.

- **ϵ-greedy Solution:** An ϵ-greedy policy provides a simple but effective way to balance this trade-off. The agent follows the greedy (exploitative) action most of the time (with probability $1 - \epsilon$), but with a small probability ϵ, it chooses an action at random from all available actions (exploration). This ensures that, over time, all actions in all states will be sampled, preventing the agent from neglecting potentially optimal actions.

9. **Question:** Explain the difference between on-policy and off-policy learning. Why is this distinction important?

   Answer:

   - **Difference:** The distinction lies in which policy is being improved relative to the policy being used to generate behavior.
     - **On-Policy:** The agent learns about and improves the *same* policy it is using to act. For example, it uses an ϵ-greedy policy to explore and collects data to make that same ϵ-greedy policy better. The goal is to find the best possible exploratory policy. SARSA is a classic example.
     - **Off-Policy:** The agent learns about a *target policy* (typically the optimal, greedy policy) using data generated by a different *behavior policy* (typically an exploratory one, like ϵ-greedy). It learns about how to act optimally while behaving sub-optimally to explore. Q-Learning is the classic example.
   - **Importance:** This distinction is important for flexibility. Off-policy methods are more powerful as they can learn from data not generated by the target policy. For instance, an off-policy agent could learn the optimal policy by observing a human expert (or another agent) or by re-using data from its own past, sub-optimal behaviors. On-policy methods are often simpler and more stable but are constrained to learning only from their own current behavior.

10. **Question:** Derive the incremental mean update rule and explain why using a constant step-size α can be advantageous over a sample average.

    Answer:

- **Derivation:** Let V_N be the mean of N returns $\{G_1, \dots, G_N\}$.

$$\begin{aligned}
V_N &= \frac{1}{N}\sum_{k=1}^{N} G_k \\
&= \frac{1}{N}\left(G_N + \sum_{k=1}^{N-1} G_k\right) \\
\text{Since } \sum_{k=1}^{N-1} G_k &= (N-1)V_{N-1}, \\
V_N &= \frac{1}{N}(G_N + (N-1)V_{N-1}) \\
&= \frac{1}{N}G_N + \frac{N-1}{N}V_{N-1} \\
&= \frac{N-1}{N}V_{N-1} + \frac{1}{N}V_{N-1} - \frac{1}{N}V_{N-1} + \frac{1}{N}G_N \\
&= V_{N-1} + \frac{1}{N}(G_N - V_{N-1})
\end{aligned}$$

  This gives the update rule: NewEstimate $\leftarrow$ OldEstimate + StepSize(Target - OldEstimate).

- **Advantage of Constant α:** When the step-size is $\alpha = 1/N$, the update rule computes a true sample average. This works well for stationary problems. However, in **non-stationary environments**, where the reward dynamics might change over time, a constant α is advantageous. A constant step-size gives more weight to recent experiences and allows the agent to gradually "forget" old, potentially outdated information. This enables the agent to adapt its value estimates if the environment changes.

7 Synthesis and The Path Forward

11. **Question:** Explain the bias-variance tradeoff as it applies to Dynamic Programming vs. Monte Carlo methods.

    Answer:

    - **Dynamic Programming (Low Variance, High Bias):**
      - **Bias:** DP methods bootstrap. Their updates are based on other estimated values (e.g., V_k is updated using V_{k-1}). If these base estimates are incorrect, the new estimate will also be incorrect, introducing a systematic error, or **bias**.
      - **Variance:** Because DP uses a perfect model to compute a full expectation over all possible outcomes, there is no randomness from sampling. Therefore, there is **zero variance** introduced from sampling.

- **Monte Carlo (Low Bias, High Variance):**
  - **Bias:** MC methods do not bootstrap. Their updates are based on the actual, full return G_t from a sample episode. This return is an unbiased sample of the true value $V^\pi(s)$. Therefore, the estimates are **unbiased**.
  - **Variance:** The return of any single episode can deviate significantly from the true expected value due to randomness in the policy and environment. This reliance on random samples introduces **high variance**. Many episodes are needed to average out this noise.

12. **Question:** Describe Temporal-Difference (TD) learning. How does it bridge the gap between DP and MC methods, and what is its main update rule?

    Answer:

    - **Description:** Temporal-Difference (TD) learning is a class of model-free reinforcement learning algorithms that represents a middle ground between Monte Carlo and Dynamic Programming.
    - **Bridging the Gap:**
      - **Like MC:** TD learning is **model-free**. It learns directly from raw experience without needing the transition probabilities P or reward function R.
      - **Like DP:** TD learning **bootstraps**. It updates its value estimates based on other learned estimates, rather than waiting for the final outcome of an episode.
    - **Main Update Rule (TD(0)):** The TD(0) update rule for the state-value function is:

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

      This update is performed at each time step. The agent takes one step, observes the reward R_{t+1} and the next state S_{t+1}, and immediately updates its estimate for the value of the state it just left, $V(S_t)$, using its current estimate for the value of the state it arrived in, $V(S_{t+1})$.

13. **Question:** Why is TD learning often more efficient than MC learning in practice?

    Answer: TD learning is often more efficient than MC for two main reasons:

    - **Lower Variance:** The MC update target is the full return G_t, which depends on a long sequence of random actions and state transitions, leading to high variance. The TD target, $R_{t+1} + \gamma V(S_{t+1})$, depends on only one random action and transition. Because the TD target is based on an existing (and hopefully stabilizing) estimate $V(S_{t+1})$, it is much less noisy than the MC target. This lower variance allows TD methods to learn with fewer samples.

- **Online Learning:** TD methods can learn online. They update their value estimates after every single time step. They do not need to wait for an episode to conclude. This makes them applicable to continuous (non-episodic) tasks and allows them to learn much faster within a single episode compared to MC, which must wait until the very end to make any updates.

14. **Question:** If you have a perfect model of an environment, why might you still choose a learning-based approach like MC or TD over a planning approach like Value Iteration?

    Answer: Even with a perfect model, a planning approach like Value Iteration might be computationally infeasible. The primary reason is the **curse of dimensionality**. The complexity of one iteration of Value Iteration is roughly $O(|A||S|^2)$. If the state space S is enormous (e.g., the number of possible board positions in Go, or a continuous state space), it becomes impossible to "sweep through" all the states to perform the Bellman updates. In such cases, a learning-based approach that *samples* from the state space might be the only practical option. The agent would interact with the (perfect) model as if it were an environment, generating sample trajectories and using methods like MC or TD with function approximation to learn a value function over the intractably large state space.

15. **Question:** Imagine you are designing an agent to learn to play a complex board game like chess. Would you use a model-based or model-free approach? Justify your choice by discussing the pros and cons of each in this context.

    Answer: For a game like chess, a **model-based approach** (or a hybrid that heavily features planning) is overwhelmingly superior.

    - **Justification:**
      (a) **A Perfect Model Exists:** The rules of chess are perfectly defined and known. The state transition function $P(s'|s,a)$ is deterministic and available: for any board position (state) and any legal move (action), the resulting board position (next state) is known with certainty. This is the ideal scenario for model-based methods.
      (b) **Power of Planning:** A model allows the agent to "look ahead" and perform deep searches into future possibilities (e.g., using alpha-beta pruning or Monte Carlo Tree Search, which is a sophisticated form of model-based planning). This ability to simulate and evaluate sequences of moves without having to learn their outcomes from scratch is incredibly powerful.
    - **Cons of a Purely Model-Free Approach:**
      - **Extreme Sample Inefficiency:** A purely model-free agent would have to play an astronomical number of games to learn the consequences of

    each move from each position through trial and error. It would be like re-discovering the rules of chess from scratch every time.

  - **Ignoring Critical Information:** It would be foolish to ignore the perfect model (the rules) when it is readily available.

- **Conclusion:** While the state space of chess is too large for a naive Value Iteration over a tabular representation, successful chess engines (like AlphaZero) use a hybrid approach. They use the model (the rules) to perform deep searches (planning) but use a neural network (a function approximator) trained via self-play (learning) to provide an evaluation function for board states, guiding the search. This combines the lookahead power of planning with the generalization power of learning.