Foundational Reinforcement Learning A Comprehensive Question Bank *Based on "A Foundational Monograph on Modern Reinforcement Learning"*

Part A: Multiple-Choice Questions (MCQ)

Section 1: The Imperative for Function Approximation

Question 1

What is the primary motivation for using function approximation in reinforcement learning?

(a) To simplify the reward function.

(b) To handle environments with a small, manageable number of states.

(c) **To overcome the "curse of dimensionality" in large state spaces. (Correct)**

(d) To ensure all updates are on-policy.

Answer

Explanation: Tabular methods, which store a value for every state, are infeasible for complex problems with vast or infinite state spaces (e.g., game of Go, robotics). This state space explosion is called the "curse of dimensionality." Function approximation is a necessity, not just an improvement, to generalize learning across these large state spaces using a manageable number of parameters.

Question 2

Which of the following best describes the "curse of dimensionality" in the context of tabular RL?

(a) The difficulty of designing reward functions for high-dimensional actions.

(b) **The exponential growth of the state space with the number of variables describing the environment. (Correct)**

(c) The high computational cost of a single policy update.

(d) The instability caused by bootstrapping in TD learning.

Answer

Explanation: The "curse of dimensionality" specifically refers to how the size of the lookup table required for tabular methods grows exponentially as more features or variables are added to define a state. This makes both the memory needed to store the table and the time needed to learn its values prohibitive.

Question 3

In the context of value function approximation, what does the parameter vector $\mathbf{w}$ (or θ) represent?

(a) The agent's policy.

(b) The reward function of the environment.

(c) A lookup table of all state-action values.

(d) **The weights of a parameterized function that approximates the true value function. (Correct)**

Answer

Explanation: Instead of storing individual values, we use a function (like a neural network) defined by a set of parameters or weights, denoted by w. The goal is to learn the optimal values for w such that the function $\hat{v}(s; w)$ accurately approximates the true value function $v_\pi(s)$.

Question 4

What is the most common objective function to be minimized when training a value function approximator?

(a) The expected total reward of a trajectory.

(b) **The Mean Squared Value Error (MSVE). (Correct)**

(c) The variance of the Monte Carlo returns.

(d) The probability of selecting a suboptimal action.

Answer

Explanation: The Mean Squared Value Error (MSVE) is the standard objective. It measures the expected squared difference between the true state values and the values predicted by the function approximator, typically weighted by how often each state is visited.

Question 5

Why is the state distribution $\mu_\pi(s)$ included in the MSVE objective function $J(w)$?

(a) To make the objective function differentiable.

(b) To simplify the calculation of the gradient.

(c) **To focus learning on states that are more frequently visited under policy π. (Correct)**

(d) To ensure the agent explores all possible states.

Answer

Explanation: Not all states are equally important. An agent will spend more time in certain states than others. By weighting the error at each state by the stationary state distribution $\mu_\pi(s)$, we prioritize making the approximation more accurate for the parts of the state space that are most relevant to the agent's behavior.

Question 6

What is Stochastic Gradient Descent (SGD) used for in this context?

(a) To select the best action in a given state.

(b) To estimate the true value function without approximation.

(c) **To iteratively update the function approximator's parameters (w) to minimize the MSVE. (Correct)**

(d) To calculate the total reward of an episode.

Answer

Explanation: SGD is the optimization algorithm used to train the function approximator. Instead of computing the true gradient over all states (which is intractable), SGD uses a single experience (or a small mini-batch) to estimate the gradient and takes a small step in the opposite direction to iteratively minimize the error.

Question 7

Generalization in function approximation is a "double-edged sword" because:

(a) **An update for one state affects many others, which allows for scaling but can also propagate errors globally. (Correct)**

(b) It requires twice the memory of tabular methods.

(c) It works for continuous states but not discrete ones.

(d) It solves the memory issue but not the learning time issue.

Answer

Explanation: Generalization is what allows learning to scale, as experience in one state provides information about similar states. However, this same property means a single bad update (due to noise or an inaccurate target) can corrupt the value estimates for many other states, a "crosstalk" effect that is a source of instability.

Section 2: Learning with Approximate Value Functions

Question 8

What is the fundamental "target problem" when applying supervised learning methods like SGD to RL?

(a) The target is a vector, which is hard to predict.

(b) **The ground-truth value $v_\pi(s)$ is unknown and must be estimated from experience. (Correct)**

(c) The target changes randomly at every time step.

(d) It is difficult to define a target for continuous action spaces.

Answer

Explanation: The SGD update rule requires a target (a "correct label") to compute the error. In RL, this true value is not provided. The agent must bootstrap or sample a target from its own stream of states, actions, and rewards. How this target is constructed is a key difference between algorithms.

Question 9

Which of the following correctly describes a property of Monte Carlo (MC) targets for value prediction?

(a) Low variance and biased.

(b) Low variance and unbiased.

(c) High variance and biased.

(d) **High variance and unbiased. (Correct)**

Answer

Explanation: The MC target is the actual return G_t from a complete episode. It's an **unbiased** sample of the true value $v_\pi(S_t)$ because, on average, it equals the true value. However, it's **high-variance** because it's a sum of many random variables (rewards, transitions), making it very noisy.

Question 10

A major disadvantage of Monte Carlo (MC) methods is that they:

(a) **Can only perform updates at the end of an episode. (Correct)**

(b) Are only suitable for deterministic environments.

(c) Cannot be used with function approximation.

(d) Introduce bias into the learning process.

Answer

Explanation: To calculate the full return G_t, MC methods must wait for the episode to terminate. This delays learning, is inefficient, and makes them inapplicable to continuing tasks that never end.

Question 11

The Temporal Difference (TD) target, $R_{t+1} + \gamma \hat{v}(S_{t+1}; w)$, introduces bias because:

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

(b) The discount factor γ is less than 1.

(c) It ignores the action taken at time t.

(d) **It depends on the current (and likely inaccurate) value estimate $\hat{v}(S_{t+1}; w)$. (Correct)**

Answer

Explanation: TD learning bootstraps—it uses the current estimate $\hat{v}(S_{t+1}; w)$ as part of its own update target. Since this estimate is not the true value $v_\pi(S_{t+1})$, the target is **biased**. Early in training, this bias can be significant.

Question 12

Compared to MC methods, TD methods are generally preferred because they have:

(a) Unbiased targets and high variance.

(b) The ability to learn without a reward signal.

(c) **Lower variance and can learn online. (Correct)**

(d) Better performance in partially observable environments.

Answer

Explanation: The TD target depends on only one reward and one transition, making it much less noisy (lower variance) than the MC target. This, combined with their ability to update after every step (online learning), generally leads to faster and more stable convergence.

Question 13

TD methods are called "semi-gradient" methods because:

(a) They only update the gradient half the time.

(b) The gradient is always positive.

(c) **They ignore the gradient of the TD target with respect to the weights w. (Correct)**

(d) They use a semi-permanent lookup table for gradients.

Answer

Explanation: A true gradient descent update would account for the fact that the target $R_{t+1} + \gamma\hat{v}(S_{t+1}; w)$ itself depends on the weights w being updated. Semi-gradient methods simplify this by ignoring the derivative of the target term, $\nabla_w \hat{v}(S_{t+1}; w)$, which is computationally convenient and works well in practice.

Question 14

In which scenario would an "action-out" architecture for $q(s, a; w)$ be most appropriate?

(a) **A video game with a small set of discrete actions (e.g., up, down, left, right). (Correct)**

(b) Controlling a robotic arm with continuous joint angles.

(c) A problem where the state is not fully observable.

(d) When using Monte Carlo learning.

Answer

Explanation: An "action-out" architecture has a separate output head for each possible action. This is efficient for problems with a small, finite number of discrete actions because a single forward pass through the network yields the Q-values for all actions, making the arg max operation easy. DQN uses this architecture.

Question 15

Which type of function approximator architecture is necessary for continuous action spaces?

(a) Action-out.

(b) **Action-in. (Correct)**

(c) Linear.

(d) Recurrent.

Answer

Explanation: With infinitely many actions in a continuous space, having a separate output for each is impossible. An "action-in" architecture is required, where both the state s and a specific action a are fed as inputs to the network, which then outputs a single Q-value for that state-action pair.

Question 16

The key difference between the on-policy SARSA update and the off-policy Q-learning update is in the:

(a) Calculation of the immediate reward R_{t+1}.

(b) Use of the discount factor γ.

(c) **Selection of the action used to form the TD target. (Correct)**

(d) Gradient of the Q-function approximator.

Answer

Explanation: The difference lies in how the value of the next state is calculated for the target. SARSA (on-policy) uses the Q-value of the action A_{t+1} that was *actually taken* by its policy. Q-learning (off-policy) uses the Q-value of the *best possible* action, $\max_a \hat{Q}(S_{t+1}, a; w)$, regardless of what the exploratory policy did.

Section 3: Deep Q-Networks (DQN)

Question 17

What is the "deadly triad" in reinforcement learning?

(a) On-policy learning, linear approximation, and high variance.

(b) High-dimensional states, continuous actions, and sparse rewards.

(c) **Function approximation, bootstrapping, and off-policy learning. (Correct)**

(d) Experience replay, fixed targets, and deep neural networks.

Answer

Explanation: The combination of these three properties is known to be a recipe for instability and divergence. Function approximation generalizes updates, bootstrapping creates targets from estimates, and off-policy learning uses a different data distribution. When combined, these can lead to compounding errors. DQN's success comes from mitigating the effects of this triad.

Question 18

What is the primary purpose of using an Experience Replay buffer in DQN?

(a) **To break temporal correlations in the training data, making samples more IID. (Correct)**

(b) To ensure the agent only learns from high-reward transitions.

(c) To decrease the amount of memory required by the algorithm.

(d) To allow the use of on-policy learning methods.

Answer

Explanation: Neural networks train best on Independent and Identically Distributed (IID) data. The sequential data collected by an RL agent is highly correlated. By storing experiences and sampling random mini-batches from this buffer, experience replay breaks these correlations, stabilizing the training process. Increased sample efficiency is a key secondary benefit.

Question 19

Experience replay increases sample efficiency because:

(a) It makes each environment interaction faster.

(b) **Each transition can be stored and reused for multiple weight updates. (Correct)**

(c) It uses a smaller neural network.

(d) It guarantees the agent will reach a terminal state.

Answer

Explanation: In online learning, an experience is used once and then discarded. With experience replay, a single transition (s, a, r, s') can be part of many different mini-batches, allowing the network to learn much more from each piece of collected data. This is crucial when interaction is costly.

Question 20

The "moving target" problem in Q-learning refers to the fact that:

(a) The environment's dynamics change over time.

(b) The agent's goal location is not fixed.

(c) The rewards are stochastic and non-stationary.

(d) **The TD target depends on the same Q-network weights that are being updated. (Correct)**

Answer

Explanation: In standard Q-learning, the target $r + \gamma \max_a \hat{Q}(s', a; w)$ changes every time the weights w are updated. This means the network is trying to converge to a target that is constantly shifting, which can lead to oscillations and instability.

Question 21

How does DQN solve the "moving target" problem?

(a) By using a very small learning rate.

(b) By updating the network only at the end of an episode.

(c) **By using a separate, periodically updated target network to calculate the TD target. (Correct)**

(d) By using Prioritized Experience Replay.

Answer

Explanation: DQN uses a second "target network" whose weights (w^-) are frozen for a large number of training steps. The TD target is calculated using this stable target network: $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; w^-)$. This provides the online network with a fixed, stationary target to learn towards, greatly improving stability.

Question 22

In the full DQN algorithm, which network is used to select the action for the ϵ-greedy policy during data collection?

(a) **The online network with weights w. (Correct)**

(b) The target network with weights w^-.

(c) A separate, randomly initialized exploration network.

(d) Both networks are used in an ensemble.

Answer

Explanation: The agent's behavior—the actions it chooses to execute in the environment—is governed by the most up-to-date network, which is the online network. This is the policy the agent is actively following. The target network is used only to compute the stable target values for the loss function.

Question 23

When are the weights of the target network updated in DQN?

(a) After every single training step.

(b) Only at the end of each episode.

(c) **Periodically (e.g., every C steps) by copying the weights from the online network. (Correct)**

(d) They are never updated; they remain at their initial random values.

Answer

Explanation: The target network weights w^- are kept frozen for a fixed number of steps (C) to ensure the target is stable. After C steps, they are updated by setting $w^- \leftarrow w$. This transfers the learned progress from the online network to the target network, which then provides a new, more accurate (but still fixed) target.

Question 24

The ablation study in Table 1 shows that using a deep network without replay or fixed targets performed worse than a linear model in some games. This suggests that:

(a) Deep networks are unsuitable for Atari games.

(b) **Naively combining deep learning and Q-learning is highly unstable. (Correct)**

(c) Linear models are generally superior for complex tasks.

(d) The reward functions for those games are poorly designed.

Answer

Explanation: This result is a key piece of evidence for the "deadly triad." Simply replacing a linear function approximator with a deep neural network in a standard Q-learning setup leads to instability that can make performance even worse. This highlights that DQN's innovations (replay and fixed targets) were essential for success, not the deep network alone.

Question 25

According to the ablation study, which single innovation had the most dramatic positive impact on performance for games like Breakout and Enduro?

(a) Adding fixed Q-targets.

(b) **Adding experience replay. (Correct)**

(c) Using a deeper neural network.

(d) Using a linear model.

Answer

Explanation: Comparing the "Deep Network (No Replay/Target)" column with the "DQN w/ Replay Only" column shows a massive performance jump (e.g., from 3 to 241 in Breakout). This suggests that breaking data correlations via experience replay was the single most critical factor for achieving stable learning in these environments.

Section 4: The Policy Gradient Theorem

Question 26

What does a policy-based reinforcement learning method learn directly?

(a) The action-value function $q(s, a)$.

(b) The state-value function $v(s)$.

(c) **A parameterized policy $\pi_\theta(a|s)$. (Correct)**

(d) The environment's transition model $p(s'|s, a)$.

Answer

Explanation: Unlike value-based methods that learn a value function and derive a policy from it, policy-based methods directly parameterize the policy itself. They learn a function $\pi_\theta(a|s)$ with parameters θ that explicitly maps states to a probability distribution over actions.

Question 27

Which of the following is a key advantage of policy gradient methods over value-based methods?

(a) They are guaranteed to converge faster.

(b) **They are more effective in continuous or high-dimensional action spaces. (Correct)**

(c) They are more sample-efficient due to experience replay.

(d) They learn a deterministic policy.

Answer

Explanation: The arg max operation over actions required by value-based methods is intractable in continuous action spaces. Policy gradient methods handle this naturally by having the policy network output the parameters (e.g., mean and variance) of a continuous probability distribution from which an action is sampled.

Question 28

Policy gradient methods are capable of learning truly stochastic policies. This is essential when:

(a) The environment is fully deterministic.

(b) The reward function is sparse.

(c) Sample efficiency is the primary concern.

(d) **The optimal policy is inherently random, e.g., in a game against an adversary. (Correct)**

Answer

Explanation: In some environments, especially competitive ones (like rock-paper-scissors), a deterministic policy can be easily exploited. The optimal strategy is to be unpredictable. Policy-based methods can learn such stochastic policies directly, whereas the stochasticity in ϵ-greedy Q-learning is just for exploration, not part of the learned optimal policy.

Question 29

The objective function $J(\theta)$ in policy gradient methods typically represents the:

(a) Mean Squared Error of the policy.

(b) Log-likelihood of the optimal trajectory.

(c) **Expected total reward of trajectories sampled from the policy π_θ. (Correct)**

(d) Variance of the policy's action distribution.

Answer

Explanation: The goal is to find the policy parameters θ that produce the best behavior. Performance is measured by the total reward the agent can expect to collect by following its policy. Therefore, the objective function $J(\theta)$ is the expected reward over the distribution of trajectories generated by π_θ.

Question 30

What is the purpose of the "log-derivative trick" in deriving the policy gradient?

(a) **To rewrite the gradient of an expectation as an expectation of a gradient, making it possible to estimate via sampling. (Correct)**

(b) To convert the objective function into a linear form.

(c) To eliminate the need for a learning rate.

(d) To prove that the policy gradient is always positive.

Answer

Explanation: The main difficulty is that the derivative ∇_θ is outside an expectation over a distribution $p_\theta(\tau)$ that depends on θ. The log-derivative trick, $\nabla_\theta p_\theta(\tau) = p_\theta(\tau)\nabla_\theta \log p_\theta(\tau)$, allows us to move the derivative inside the expectation. This transforms the gradient into a form that can be estimated with Monte Carlo sampling from trajectories.

Question 31

In the final expression for the policy gradient, $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right) r(\tau)\right]$, the term $r(\tau)$ acts as a:

(a) Learning rate.

(b) Normalization factor.

(c) **Scalar weight that scales the update for the entire trajectory. (Correct)**

(d) Policy parameter vector.

Answer

Explanation: The term $r(\tau)$ is the total reward for the trajectory. It is a single scalar number. If this number is large and positive, the gradient update strongly increases the probabilities of all actions taken. If it is negative or small, the update is weak or pushes the probabilities down. It "weights" the significance of the trajectory's updates.

Question 32

The score function, $\nabla_\theta \log \pi_\theta(a_t|s_t)$, intuitively points in the direction that:

(a) Maximizes the total reward of the trajectory.

(b) **Most increases the probability of taking action a_t in state s_t. (Correct)**

(c) Minimizes the variance of the action distribution.

(d) Moves the policy parameters towards zero.

Answer

Explanation: This term comes from the derivative of the log-probability of an action. The gradient of a function points in the direction of steepest ascent. Therefore, this vector indicates how we should change the parameters θ to make the action a_t more likely when the agent is in state s_t.

Section 5: REINFORCE

Question 33

The REINFORCE algorithm is considered a Monte Carlo method because:

(a) It uses a replay buffer to store experiences.

(b) **It performs parameter updates only after collecting a full episode (trajectory). (Correct)**

(c) It uses bootstrapping to estimate future rewards.

(d) Its gradient estimate is always biased.

Answer

Explanation: The defining feature of Monte Carlo RL methods is that they learn from complete episodes. REINFORCE must wait until a trajectory is complete to calculate the total return $r(\tau)$, which is needed for the update. It does not learn incrementally from single steps.

Question 34

What is the most significant weakness of the REINFORCE algorithm?

(a) Its inability to handle continuous action spaces.

(b) Its bias in the gradient estimate.

(c) Its high memory consumption.

(d) **Its high variance in the gradient estimate, leading to slow and unstable training. (Correct)**

Answer

Explanation: Because the update for every action in an episode is scaled by the return of the *entire* episode, the learning signal is extremely noisy (high-variance). A single random event can dramatically change the total return, causing good actions to be penalized or bad actions to be reinforced. This makes convergence very slow.

Question 35

REINFORCE is considered sample-inefficient because:

(a) It requires a large neural network.

(b) The policy updates are very slow to compute.

(c) **It is an on-policy algorithm and must discard all experience after each policy update. (Correct)**

(d) It can only learn from expert demonstrations.

Answer

Explanation: REINFORCE is on-policy, meaning the data used for updates must be generated by the most recent policy π_θ. Once θ is updated, all previously collected data becomes "stale" and cannot be reused. This constant need to generate fresh data makes it much less efficient than off-policy methods like DQN that use experience replay.

Question 36

An improved variant of REINFORCE weights the score function for an action a_t by the return-from-time-t (G_t) instead of the total return. This is done to:

(a) Simplify the calculation.

(b) **Reduce variance by only crediting actions with the rewards that follow them. (Correct)**

(c) Remove the bias from the gradient estimate.

(d) Allow for off-policy learning.

Answer

Explanation: This change incorporates the idea of causality: an action at time t cannot influence past rewards. By using the "reward-to-go" $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$, the credit assignment becomes more precise. This doesn't change the theoretical expected gradient but significantly reduces the sample-to-sample variance, leading to more stable learning.

Question 37

The gradient ascent update rule in REINFORCE, $\theta \leftarrow \theta + \alpha G_i \nabla_\theta \log \pi_\theta(a_t|s_t)$, means that:

(a) **If the return G_i is positive, the policy is changed to make action a_t more likely. (Correct)**

(b) The learning rate α is adjusted based on the return G_i.

(c) If the return G_i is positive, the policy is changed to make action a_t less likely.

(d) The update is only performed if the return G_i is greater than a certain threshold.

Answer

Explanation: The algorithm performs gradient ascent (hence the '+' sign). The gradient term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ points in the direction to increase the probability of action a_t. This is scaled by the return G_i. If G_i is positive, the update moves θ in that direction. If G_i were negative, it would move in the opposite direction, making the action less likely.

Question 38

How is the expectation in the policy gradient theorem estimated in the REINFORCE algorithm?

(a) Using a Taylor series expansion.

(b) Analytically solving the integral.

(c) Using a separate critic network.

(d) **By averaging the gradient contributions from one or more sampled trajectories. (Correct)**

Answer

Explanation: The expectation $\mathbb{E}_{\tau \sim p_\theta(\tau)}[\cdot]$ is intractable to compute directly. REINFORCE uses the standard Monte Carlo approach: generate a sample (or a batch of samples) from the distribution and use the sample mean as an estimate of the expectation.

Section 6: Value-Based vs. Policy-Based Methods

Question 39

The core distinction between value-based and policy-based methods is learning:

(a) A linear vs. a non-linear function.

(b) On-policy vs. off-policy.

(c) Deterministic vs. stochastic transitions.

(d) **"How good" an action is vs. "what to do" in a state. (Correct)**

Answer

Explanation: Value-based methods (like DQN) focus on learning an action-value function, $q(s, a)$, which answers "how good is it to take action a in state s?". The policy is then implicit (e.g., greedy). Policy-based methods (like REINFORCE) directly learn a policy, $\pi(a|s)$, which answers "what action should I take (or what is the probability of taking each action) in state s?".

Question 40

Which paradigm is generally considered more sample-efficient and why?

(a) **Value-based methods, because they are often off-policy and can use experience replay. (Correct)**

(b) Policy-based methods, because their gradient updates are more stable.

(c) Value-based methods, because they learn a simpler function.

(d) Policy-based methods, because they can handle continuous action spaces.

Answer

Explanation: The ability to reuse past data via an experience replay buffer gives off-policy value-based methods like DQN a significant advantage in sample efficiency. On-policy methods like REINFORCE must discard data after every update, requiring a constant stream of new interactions.

Question 41

Policy-based methods often exhibit better convergence properties because:

(a) Their loss function is always convex.

(b) They learn faster than value-based methods.

(c) **They perform smoother policy updates compared to the discontinuous changes possible in value-based methods. (Correct)**

(d) They are not affected by the "deadly triad".

Answer

Explanation: In policy-based methods, a small change in parameters θ leads to a small, smooth change in action probabilities. In value-based methods, a tiny change in a Q-value can cause the $\arg\max$ action to abruptly switch from one action to another, representing a large, discontinuous jump in the policy. This can lead to oscillations and instability.

Question 42

According to the text, the optimal policy function might be simpler to represent with a neural network than the optimal value function. What does this suggest?

(a) Policy-based methods should always be preferred.

(b) Value functions are not useful for reinforcement learning.

(c) **The empirical stability of policy-based methods might stem from them learning a fundamentally less complex function. (Correct)**

(d) All reinforcement learning problems can be solved with simple policies.

Answer

Explanation: The text mentions that even when the optimal policy is simple, the corresponding optimal value function can be highly complex and non-smooth. This "representational complexity" gap suggests that policy-based methods might be more stable and successful in some cases because the function they are trying to learn is inherently easier for a neural network to approximate.

Question 43

DQN learns a deterministic policy, with randomness added for exploration via ϵ-greedy. This means:

(a) It cannot solve problems with deterministic transitions.

(b) It will always perform worse than REINFORCE.

(c) **It cannot learn an optimal policy that is itself stochastic. (Correct)**

(d) Exploration is not necessary for DQN.

Answer

Explanation: The policy that DQN learns is to always take the action with the maximum Q-value. The randomness from ϵ-greedy is an external mechanism to ensure exploration. If the truly optimal policy for a task requires randomized action selection (e.g., in rock-paper-scissors), DQN cannot learn it. Policy-based methods, which naturally output a probability distribution, can.

Section 7: On-Policy vs. Off-Policy Learning

Question 44

What is the definition of an on-policy learning algorithm?

(a) It can only be used for control, not prediction.

(b) The behavior policy and target policy are different.

(c) **The policy being learned is the same as the policy used to generate data. (Correct)**

(d) It does not use bootstrapping.

Answer

Explanation: In on-policy learning, the behavior policy (used for data collection) and the target policy (being improved) are one and the same ($b = \pi$). The agent learns about the policy it is currently executing. REINFORCE and SARSA are canonical examples.

Question 45

What is the defining characteristic of an off-policy learning algorithm?

(a) It must be a value-based method.

(b) **It can learn about a target policy using data generated by a different behavior policy. (Correct)**

(c) It learns a separate model of the environment.

(d) It is guaranteed to be more stable than on-policy methods.

Answer

Explanation: In off-policy learning, the agent can have a different target policy (π) and behavior policy (b). For example, Q-learning learns about the optimal greedy policy (π) while behaving according to an exploratory ϵ-greedy policy (b).

Question 46

The ability to use experience replay is a direct consequence of an algorithm being:

(a) On-policy.

(b) **Off-policy. (Correct)**

(c) Value-based.

(d) Policy-based.

Answer

Explanation: Experience replay stores transitions generated by many past policies. An algorithm can only learn from this "old" data if it is off-policy—that is, if it can learn about a target policy (e.g., the current optimal one) using data generated by other, outdated policies. On-policy methods cannot do this.

Question 47

Which of the following pairs correctly identifies an on-policy and an off-policy algorithm?

(a) On-policy: Q-Learning, Off-policy: SARSA.

(b) **On-policy: REINFORCE, Off-policy: DQN. (Correct)**

(c) On-policy: DQN, Off-policy: REINFORCE.

(d) On-policy: SARSA, Off-policy: REINFORCE.

Answer

Explanation: REINFORCE updates its policy π_θ using data generated from that same policy, making it on-policy. DQN (which is based on Q-learning) learns about the greedy policy while behaving ϵ-greedily and using a replay buffer of old data, making it off-policy.

Question 48

A key advantage of off-policy learning is the decoupling of exploration and control. This means:

(a) The agent does not need to explore.

(b) **The agent can use a highly exploratory policy to collect data while learning about a purely optimal one. (Correct)**

(c) The learning process is completely independent of the actions taken.

(d) Control of the environment is passed to a separate module.

Answer

Explanation: In off-policy learning, the behavior policy can be made very random to ensure it thoroughly explores the environment and gathers diverse data. At the same time, the target policy can be purely greedy and optimal, without being compromised by the need to explore. In on-policy methods, the single policy must balance both exploration and exploitation.

Section 8: Learning from Data: Reinforcement vs. Imitation

Question 49

Behavioral Cloning (BC) frames the problem of learning from expert demonstrations as a:

(a) Reinforcement learning problem.

(b) Unsupervised learning problem.

(c) **Supervised learning problem. (Correct)**

(d) Trajectory optimization problem.

Answer

Explanation: Behavioral Cloning uses a dataset of expert (state, action) pairs. The states are treated as inputs (features) and the expert's actions are the target outputs (labels). The goal is to train a model that mimics this mapping, which is a standard supervised learning setup.

Question 50

The update rule for Behavioral Cloning is mathematically equivalent to the policy gradient update from REINFORCE if:

(a) The discount factor is set to 0.

(b) **The reward is assumed to be +1 for every action taken by the expert. (Correct)**

(c) The learning rate is very high.

(d) The policy is deterministic.

Answer

Explanation: The text shows the two update rules side-by-side. The Behavioral Cloning update lacks the reward weighting term (G_t or $r(\tau)$) present in the REINFORCE update. This is equivalent to having a reward of +1 for every state-action pair in the expert data, which means every expert action is reinforced equally, regardless of its outcome.

Question 51

A major weakness of Behavioral Cloning related to this equivalence is that it:

(a) Is too slow to train.

(b) **Cannot distinguish between optimal and suboptimal actions within the expert data. (Correct)**

(c) Can only be used for discrete actions.

(d) Requires an accurate model of the environment.

Answer

Explanation: Because Behavioral Cloning effectively treats every demonstrated action as "correct" (reward = +1), it has no mechanism to learn that some of the expert's actions might have been better than others, or even mistakes. It simply tries to mimic everything, without an external, evaluative signal like a reward to guide it toward better outcomes.

Question 52

In which of the following scenarios is imitation learning like Behavioral Cloning most suitable?

(a) When the goal is to discover a policy that is better than any human expert.

(b) **When interacting with the environment is expensive or dangerous, but expert data is available. (Correct)**

(c) When the reward function is easy to define and provides dense feedback.

(d) When sample efficiency is not a concern.

Answer

Explanation: Imitation learning shines when online interaction is not feasible (e.g., training a self-driving car in the real world initially). If we have a dataset of safe, expert driving behavior, we can pre-train a policy without risky trial-and-error. RL is better when such interaction is cheap and the goal is to exceed expert performance.

Question 53

What is the "distribution shift" problem in Behavioral Cloning?

(a) The expert's behavior changes over time.

(b) The policy is trained on data from one environment and tested on another.

(c) **If the agent makes a mistake and enters a state not seen in the expert data, its behavior can become unpredictably bad. (Correct)**

(d) The reward distribution of the environment is non-stationary.

Answer

Explanation: The agent is trained only on the distribution of states visited by the expert. If the agent makes one small error, it may find itself in a state that the expert never encountered. Since it wasn't trained on data from this part of the state space, its policy may be completely wrong, leading to more errors that take it further away from the expert distribution (compounding errors).

Section 9: Hybrid Methods and Conclusion

Question 54

What is the primary motivation for Actor-Critic methods?

(a) **To combine the strengths of value-based and policy-based methods. (Correct)**

(b) To create a completely model-based algorithm.

(c) To eliminate the need for a discount factor.

(d) To provide a simpler alternative to REINFORCE.

Answer

Explanation: Actor-Critic methods aim to get the best of both worlds: the stability and continuous action space capabilities of policy-based methods (the Actor) and the lower variance and improved sample efficiency from bootstrapping value functions (the Critic).

Question 55

In an Actor-Critic architecture, what is the role of the "Actor"?

(a) To estimate the value function.

(b) **To control the agent's behavior by learning a policy $\pi_\theta(a|s)$. (Correct)**

(c) To model the environment's dynamics.

(d) To store past experiences in a replay buffer.

Answer

Explanation: The Actor is the policy. It is a parameterized function $\pi_\theta(a|s)$ that takes the current state and decides which action to take (or provides a distribution over actions). It's the component that "acts."

Question 56

In an Actor-Critic architecture, what is the role of the "Critic"?

(a) **To evaluate the Actor's actions by learning a value function (e.g., $\hat{v}(s; w)$). (Correct)**

(b) To determine the agent's policy.

(c) To provide the reward signal from the environment.

(d) To decide when the Actor's policy should be updated.

Answer

Explanation: The Critic does not make decisions. Its job is to learn a value function (either a state-value function $V(s)$ or an action-value function $Q(s, a)$) that "critiques" the actions taken by the actor. This critique (e.g., the TD error) is then used as a low-variance learning signal to update the Actor's policy.

Question 57

How does the Critic help reduce the variance of the policy gradient update in REINFORCE?

(a) By making the policy deterministic.

(b) **By replacing the high-variance Monte Carlo return G_t with a lower-variance TD error from the learned value function. (Correct)**

(c) By using a larger batch size for updates.

(d) By ensuring the agent only visits high-reward states.

Answer

Explanation: The core problem of REINFORCE is the high variance of the MC return G_t. Actor-Critic methods replace this noisy, full-episode return with a bootstrapped estimate of value from the Critic. The TD error, for example, is a much lower-variance (though biased) signal that is used to guide the Actor's update, leading to more stable learning.

Section 10: Mixed Review Questions

Question 58

An RL agent interacting with an environment with 10^{170} states (like Go) makes tabular methods impossible due to:

(a) **Prohibitive memory requirements and the astronomical time needed to visit every state. (Correct)**

(b) The lack of a clear reward signal.

(c) The discrete nature of the action space.

(d) The difficulty of designing a good exploration strategy.

Answer

Explanation: This question goes back to the core motivation in Section 1. A table with 10^{170} entries is impossible to store in any computer (memory) and an agent could never visit even a tiny fraction of these states in the lifetime of the universe (learning time).

Question 59

An update for an "action-in" $q(s, a; w)$ approximator requires:

(a) A single forward pass to get all Q-values.

(b) **The state and a specific action to be provided as input to the network. (Correct)**

(c) A discrete action space.

(d) A value function for each action.

Answer

Explanation: Recalling from Section 2, the "action-in" architecture is designed to handle cases (especially continuous action spaces) where the action itself is part of the input. To get a Q-value, you must provide both the state and the action you want to evaluate.

Question 60

DQN uses an ϵ-greedy policy for action selection. What does this mean?

(a) The agent always chooses the action with the highest predicted Q-value.

(b) **With probability ϵ, the agent chooses a random action; otherwise, it chooses the action with the highest Q-value. (Correct)**

(c) The agent's learning rate is annealed by a factor of ϵ.

(d) The agent selects actions from a Gaussian distribution with variance ϵ.

Answer

Explanation: This is the definition of ϵ-greedy exploration. It's a simple and effective strategy to balance exploration (taking random actions to discover new things) and exploitation (taking the best-known action to maximize reward).

Question 61

Which algorithm would be most suitable for controlling a helicopter, a task with a continuous state and action space?

(a) Tabular Q-learning.

(b) Standard DQN.

(c) **A policy gradient method like REINFORCE (or more advanced Actor-Critic methods). (Correct)**

(d) Monte Carlo Prediction.

Answer

Explanation: The continuous action space makes standard DQN (which requires an $\arg\max$ over discrete actions) unsuitable. Policy gradient methods are the natural choice as they can output parameters for a continuous action distribution.

Question 62

If you train a REINFORCE agent and notice that its performance fluctuates wildly between episodes, the most likely cause is:

(a) Bias in the gradient estimate.

(b) Using a fixed target network.

(c) **High variance from the Monte Carlo returns. (Correct)**

(d) Insufficient exploration.

Answer

Explanation: Wild fluctuations in performance are the classic symptom of high-variance gradient updates. A lucky episode can cause performance to spike, and an unlucky one can cause it to crash. This is the main weakness of REINFORCE.

Question 63

What allows Q-learning to be off-policy?

(a) The use of a function approximator.

(b) The fact that it updates after every step.

(c) **The max operator in its update rule, which considers the value of the optimal action, not the one that was actually taken. (Correct)**

(d) The use of a discount factor γ.

Answer

Explanation: The target in Q-learning, $r+\gamma \max_a \hat{Q}(s', a; w)$, estimates the return assuming the agent will follow the optimal (greedy) policy from state s' onwards. It doesn't matter what action the exploratory behavior policy actually took to get from s' to the next state. This is what decouples the target policy (greedy) from the behavior policy (ϵ-greedy).

Question 64

The term "bootstrapping" in RL refers to:

(a) Re-sampling data with replacement to estimate statistics.

(b) Starting the agent in a random state.

(c) **Updating a value estimate based on other, subsequent value estimates. (Correct)**

(d) Storing experience in a large memory buffer.

Answer

Explanation: Bootstrapping describes the process where an estimate is updated using other estimates. TD methods like Q-learning do this, as the target for $V(S_t)$ includes the estimated value $V(S_{t+1})$. MC methods do not bootstrap, as their target is the actual, full return.

Question 65

According to the ablation study for DQN, which combination of components yields the best performance on average?

(a) Experience Replay only.

(b) Fixed Q-Targets only.

(c) A deep network with neither component.

(d) **The full DQN with both Experience Replay and Fixed Q-Targets. (Correct)**

Answer

Explanation: The final column in Table 1, "Full DQN," shows the highest scores for almost every game. This demonstrates a synergistic effect where stabilizing the data with replay and stabilizing the target with a fixed network work together to produce the most robust and effective algorithm.

Question 66

Which algorithm would be theoretically unable to solve a game like Rock-Paper-Scissors optimally against a perfect opponent?

(a) REINFORCE.

(b) An Actor-Critic method.

(c) **DQN. (Correct)**

(d) A general policy gradient method.

Answer

Explanation: The optimal policy in Rock-Paper-Scissors is stochastic (play each with 1/3 probability). DQN learns a deterministic policy (arg max of Q-values). While its ϵ-greedy behavior is random, the policy it is *learning* is not, and would be easily exploited. Policy-based methods can learn a truly stochastic optimal policy.

Question 67

In policy gradient methods, what would happen if you used a constant value (e.g., $c = 100$) instead of the trajectory reward $r(\tau)$ in the update rule?

(a) **It would push the policy to maximize the log-probabilities of all actions, without regard for outcomes. (Correct)**

(b) The policy would converge to a random policy.

(c) It would be equivalent to Q-learning.

(d) The variance of the gradient would decrease.

Answer

Explanation: The reward term provides the crucial evaluative signal. Without it (or with a constant positive reward), the algorithm would simply try to make every action it takes more likely, as there's no distinction between good and bad trajectories. This is similar to the issue in behavioral cloning.

Question 68

What is the main advantage of TD learning over MC learning that makes it suitable for continuing tasks?

(a) It has a biased target.

(b) It can only be used with function approximation.

(c) **It can learn online and does not require waiting for an episode to terminate. (Correct)**

(d) It has higher variance.

Answer

Explanation: Continuing tasks have no terminal state. MC methods, which require a final return G_t, are inapplicable. TD methods can update after a single time step using the one-step-ahead reward and value estimate, making them ideal for online learning in both episodic and continuing tasks.

Question 69

An "action-out" architecture in DQN is computationally advantageous for discrete action spaces because:

(a) **It computes the Q-values for all actions in a single forward pass, making the** $\arg\max$ **efficient. (Correct)**

(b) It requires fewer parameters than an "action-in" architecture.

(c) It can be trained without a replay buffer.

(d) The backpropagation step is simpler.

Answer

Explanation: To find the best action, the agent needs to compute $\max_a Q(s, a; w)$. With an "action-out" architecture, the network takes the state s as input and outputs a vector of Q-values, one for each action. A single forward pass is all that's needed to get all the values required for the maximization.

Question 70

The "crosstalk" between states via shared parameters in function approximation means that:

(a) States are not independent and identically distributed.

(b) The reward function is corrupted.

(c) **An update based on one state's experience will alter the predicted values for all other states. (Correct)**

(d) Communication between different agents is required.

Answer

Explanation: In a tabular method, changing the value of $v(s_1)$ has no effect on $v(s_2)$. With a function approximator defined by weights w, updating w based on an experience at s_1 changes the function itself. This alters the predicted values $\hat{v}(s; w)$ for *all* states s, not just s_1. This is the mechanism of generalization and a potential source of instability.

Question 71

In the SARSA acronym, what does the final 'A' stand for?

(a) The reward received after taking the next action.

(b) The approximated Q-value.

(c) **The specific action, A_{t+1}, that was chosen in the next state, S_{t+1}. (Correct)**

(d) The advantage function.

Answer

Explanation: SARSA stands for (S, A, R, S', A'). The update rule for the transition from S_t to S_{t+1} depends on the action A_t taken, the reward R_{t+1} received, the next state S_{t+1} reached, AND the next action A_{t+1} that the agent actually chooses in state S_{t+1}. This makes it on-policy.

Question 72

If a policy gradient agent is in a state s and takes an action a that leads to a very large negative reward, the update will adjust θ to:

(a) Make action a much more likely in state s.

(b) **Make action a less likely in state s. (Correct)**

(c) Leave the probability of action a unchanged.

(d) Increase the learning rate.

Answer

Explanation: The update is proportional to $\nabla_\theta J(\theta) \approx G \cdot \nabla_\theta \log \pi_\theta(a|s)$. If the return G is a large negative number, the update $\theta \leftarrow \theta + \alpha G \nabla_\theta \log \pi_\theta(a|s)$ will move the parameters θ in the *opposite* direction of the score function, effectively decreasing the probability of taking action a in state s in the future.

Question 73

DQN can be viewed as structuring the RL problem into a series of:

(a) **Standard supervised learning problems. (Correct)**

(b) Unsupervised clustering problems.

(c) On-policy evaluation problems.

(d) Open-loop control problems.

Answer

Explanation: The text makes this insightful point in section 3.2. By using a replay buffer and fixed targets, at each update step DQN is solving a straightforward supervised learning task: given a mini-batch of inputs (states), train the network to predict a batch of target outputs (the stable y_i values).

Question 74

The policy gradient theorem's derivation relies on the assumption that the environment dynamics $p(s'|s, a)$ are:

(a) Differentiable with respect to θ.

(b) **Independent of the policy parameters θ. (Correct)**

(c) Deterministic.

(d) Unknown to the agent.

Answer

Explanation: During the derivation of $\nabla_\theta \log p_\theta(\tau)$, we differentiate a sum of terms. The gradient of the terms related to the environment dynamics, $\log p(s'|s, a)$, is zero because the environment's physics are assumed not to change when the agent updates its policy parameters θ.

Question 75

Which algorithm inherently trades higher bias for lower variance compared to its counterpart?

(a) Monte Carlo (MC) Learning.

(b) **Temporal Difference (TD) Learning. (Correct)**

(c) Behavioral Cloning.

(d) REINFORCE.

Answer

Explanation: This is the fundamental trade-off between MC and TD. MC uses an unbiased but high-variance target (the full return). TD introduces bias by bootstrapping from the current value estimate, but in return it achieves a much lower-variance target, which often leads to faster convergence.

Question 76

The "implicit policy" in a value-based method like DQN refers to:

(a) **The policy of choosing the action that maximizes the learned Q-function. (Correct)**

(b) A separate neural network that learns the policy.

(c) The use of an ϵ-greedy exploration strategy.

(d) A policy that is hidden from the environment.

Answer

Explanation: DQN does not have an explicit representation of its policy. The policy is implicitly defined by the learned Q-values; the agent's behavior is derived from them by taking the 'argmax'.

Question 77

Why can a small change to a Q-value in DQN lead to a large change in policy?

(a) Because the network has many layers.

(b) **Because the $\arg\max$ operation is discontinuous; a tiny change can make a different action have the highest value. (Correct)**

(c) Because of the experience replay buffer.

(d) Because the learning rate is too high.

Answer

Explanation: Imagine two actions have Q-values of 4.1 and 4.2. The policy chooses the second action. A small update might change them to 4.3 and 4.25. Now the policy abruptly switches to the first action. This discontinuous jump is a source of instability that smoother policy gradient updates avoid.

Question 78

The central challenge that Actor-Critic methods attempt to solve is the:

(a) Instability of the deadly triad.

(b) **High variance of policy gradient estimates. (Correct)**

(c) Problem of credit assignment in episodic tasks.

(d) Distribution shift in imitation learning.

Answer

Explanation: While related to overall stability, the most direct problem Actor-Critic methods address is the high variance from the Monte Carlo returns used in simpler policy gradient methods like REINFORCE. The Critic's role is to provide a low-variance learning signal (like the TD error) to guide the Actor.

Question 79

If you have a large dataset of an expert playing a game but cannot interact with the game engine, which method is your only option?

(a) DQN.

(b) REINFORCE.

(c) **Behavioral Cloning. (Correct)**

(d) SARSA.

Answer

Explanation: All standard RL methods (DQN, REINFORCE, SARSA) require active interaction with the environment to collect new data through trial and error. Behavioral Cloning is a form of imitation learning that works purely offline from a fixed dataset of demonstrations.

Question 80

The successful application of RL to a problem with a state space of 10^{74000} (from a 128x128 RGB image) proves that:

(a) All states were visited during training.

(b) The reward function for vision tasks is simple.

(c) **The learned function approximator successfully generalized from seen states to unseen ones. (Correct)**

(d) Tabular methods are superior for vision.

Answer

Explanation: An agent could never see more than a minuscule fraction of 10^{74000} states. Success on such a task is definitive proof of generalization: the function approximator (e.g., a convolutional neural network) learned underlying features and patterns that allowed it to make good decisions even in novel image states it had never encountered before. This is the entire point of function approximation.

Part B: Essay and Explanatory Questions

Essay Question 1

Explain in detail the "curse of dimensionality" and why it makes tabular reinforcement learning methods intractable for most real-world problems. Provide two distinct consequences of this problem as discussed in the text.

Answer

Explanation: The "curse of dimensionality" in reinforcement learning refers to the exponential growth of the number of possible states as we add more variables or features to describe the environment. For tabular methods like Q-learning or SARSA, which require storing a value for every single state (or state-action pair) in a lookup table, this growth presents an insurmountable barrier for any problem of meaningful complexity.
Consider an environment described by d variables, where each can take on k values. The total number of states is k^d. If we simply add one more variable, the state space multiplies by a factor of k. This exponential scaling is what makes tabular methods collapse. For example, a simple 128x128 pixel RGB image has $(256)^{128\times128\times3} \approx 10^{74000}$ possible states, a number far greater than the number of atoms in the known universe. Storing a table of this size is a physical impossibility.
The text highlights two primary consequences of this state space explosion:

1. **Prohibitive Memory Requirements:** The most direct consequence is the memory required to store the value table. As the number of states explodes into the billions, trillions, or (as in the case of images) astronomically large numbers, the RAM required to hold the table exceeds the capacity of any computer. For continuous state spaces, the number of states is infinite, making a lookup table conceptually impossible from the start.

2. **Prohibitive Learning Time (Sample Complexity):** Even more critical than the memory issue is the time required to learn the values. To get a reliable estimate for the value of a single state, the agent must visit it multiple times. To learn a complete value function, the agent would need to explore the entire state space, visiting every state-action pair many times. For a state space of 10^{20} (like Backgammon), this is completely infeasible. The agent would never gather enough experience in its lifetime to learn meaningful values for more than a tiny fraction of the states.

Because of these consequences, the monograph argues that function approximation is not merely an alternative approach but a *fundamental necessity* to apply reinforcement learning to real-world problems. We must shift the goal from storing an exact value for every state to *approximating* the underlying value function with a compact, parameterized model.

Essay Question 2

Compare and contrast Monte Carlo (MC) and Temporal Difference (TD) learning for the prediction problem. Discuss their targets, the bias-variance trade-off, and their applicability to different types of tasks.

Answer

Explanation: Monte Carlo (MC) and Temporal Difference (TD) are two primary methods for learning value functions from experience. They differ fundamentally in how they construct the "target" for their updates, which leads to a critical trade-off between bias and variance.

1. Target Construction:

- **MC Target:** The MC target is the **full return**, $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$. It is the actual, observed sum of discounted rewards from time step t until the end of the episode. To compute this, the algorithm must wait for the entire episode to finish.
- **TD Target:** The TD target is a **one-step bootstrapped estimate**, $Y_t = R_{t+1} + \gamma \hat{v}(S_{t+1}; w)$. It consists of the immediate reward, R_{t+1}, plus the discounted *current estimate* of the next state's value. It does not wait for the episode to end, instead using its own (potentially flawed) knowledge to "fill in" the rest of the return.

2. The Bias-Variance Trade-off:

- **MC (Unbiased, High-Variance):** The MC target G_t is an **unbiased** estimate of the true value $v_\pi(S_t)$ because, by definition, the expected value of the sample return is the true value. However, G_t is a sum of many random variables (all future rewards and transitions). A single stochastic event can drastically change its value, making it extremely noisy and **high-variance**. This leads to unstable updates and slow convergence.
- **TD (Biased, Lower-Variance):** The TD target is **biased** because it depends on the current value estimate $\hat{v}(S_{t+1}; w)$, which is not the true value, especially early in training. However, the target only depends on one random reward (R_{t+1}) and one random transition. This makes it substantially less noisy and **lower-variance** than the MC target, which generally leads to faster and more stable learning. TD learning trades a little bias for a large reduction in variance.

3. Applicability:

- **MC Methods** are restricted to **episodic tasks** (tasks that have a defined end point). Because they must wait for a final return to be calculated, they cannot be used for continuing tasks that go on forever. Furthermore, their learning is delayed and often inefficient.
- **TD Methods** are far more flexible. They can perform updates after every single time step, allowing for **online and incremental learning**. This makes them much more sample-efficient and applicable to both **episodic and continuing tasks**.

In summary, while MC methods have the appealing property of being unbiased, the practical advantages of TD learning—lower variance and the ability to learn online from incomplete sequences—have made it the dominant paradigm in modern reinforcement learning, forming the basis for algorithms like Q-learning and DQN.

Essay Question 3

DQN introduced two key innovations to stabilize the combination of off-policy TD learning and deep neural networks: Experience Replay and Fixed Q-Targets. Explain the problem that each innovation solves and describe how it works.

Answer

Explanation: The naive combination of off-policy TD learning (like Q-learning) with non-linear function approximators (like deep neural networks) is notoriously unstable. This instability stems from two primary issues arising from the "deadly triad." DQN's major contribution was introducing two mechanisms to solve these specific problems: Experience Replay and Fixed Q-Targets.

1. Innovation: Experience Replay

- **Problem Solved: Correlated Data and Non-IID Samples.** When an RL agent interacts with an environment, the sequence of experiences $(s_t, a_t, r_t, s_{t+1}), (s_{t+1}, \dots), \dots$ is highly correlated. State s_{t+1} is directly dependent on s_t. Training a neural network on such a sequential, correlated stream of data violates the fundamental assumption of Independent and Identically Distributed (IID) samples, which is crucial for stable stochastic gradient descent. This leads to inefficient learning and can cause the network's parameters to oscillate or diverge.
- **How it Works:** Experience replay introduces a large memory buffer (a replay buffer) that stores past transitions. Instead of training on the most recent transition, the algorithm samples a *random mini-batch* of transitions from this buffer.
  - **Breaks Correlations:** By sampling randomly, the transitions in a mini-batch come from many different time steps and past policies. This shuffles the experience, breaking the temporal correlations and making the training data for each update step much closer to being IID. This is the primary reason for the stabilization effect.
  - **Increases Sample Efficiency:** Each experience can be stored and reused for multiple weight updates, allowing the agent to learn more from each interaction.

2. Innovation: Fixed Q-Targets

- **Problem Solved: Non-Stationary or "Moving" Targets.** In standard Q-learning, the update target is $y_i = r_i + \gamma \max_{a'} Q(s', a'; w)$. The issue is that the weights w used to calculate the target are the *same weights* that are being updated in the gradient descent step. This creates a "moving target" problem: the network is trying to chase a target that is simultaneously moving with each update. This self-referential process can lead to oscillations and divergence.
- **How it Works:** DQN introduces a second neural network, the **target network**, with parameters w^-.
  - **Separate Networks:** An *online network* (with weights w) is updated at every step and is used for action selection. A *target network* (with weights w^-) is a clone of the online network, but its weights are kept frozen.
  - **Stable Target Calculation:** The TD target is calculated using the frozen target network: $y_i = r_i + \gamma \max_{a'} Q(s', a'; w^-)$. Because w^- is fixed, the target y_i is stable and stationary for many updates.
  - **Periodic Updates:** Every C steps, the weights of the target network are updated by copying the weights from the online network ($w^- \leftarrow w$). This provides the online network with a consistent objective for a fixed

Essay Question 4

Derive the Policy Gradient Theorem, starting from the objective function $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]$. Explain the critical role of the "log-derivative trick" in this derivation and provide an intuitive interpretation of the final gradient expression.

Answer

Explanation: The Policy Gradient Theorem provides a way to compute the gradient of the expected trajectory reward with respect to the policy parameters, θ, enabling direct optimization via gradient ascent.

1. Objective Function and Initial Gradient The objective is to maximize the expected total reward over trajectories τ generated by the policy π_θ. The distribution of trajectories is denoted $p_\theta(\tau)$.

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau$$

Taking the gradient with respect to θ:

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau$$

The problem here is that this is not an expectation, so we cannot estimate it with Monte Carlo sampling. We are stuck unless we can transform $\nabla_\theta p_\theta(\tau)$ back into a form involving $p_\theta(\tau)$.

2. The Log-Derivative Trick This is the crucial step. We use the identity from calculus for the derivative of a logarithm: $\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$. Rearranging this gives:

$$\nabla_x f(x) = f(x) \nabla_x \log f(x)$$

We can apply this trick to our term $\nabla_\theta p_\theta(\tau)$:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$$

3. Rewriting the Gradient as an Expectation Substituting this back into our gradient expression:

$$\nabla_\theta J(\theta) = \int (p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)) r(\tau) d\tau$$

We can now group the terms to form an expectation again:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[(\nabla_\theta \log p_\theta(\tau)) r(\tau) \right]$$

This is powerful because the expression inside the expectation can now be computed for any sampled trajectory, making the gradient estimable.

4. Simplifying the Gradient Term The final step is to simplify $\nabla_\theta \log p_\theta(\tau)$. The probability of a trajectory is:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Taking the log:

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t, a_t)$$

When we take the gradient ∇_θ, the terms for the initial state distribution and the environment dynamics drop out, as they do not depend on the policy parameters θ. We are left with:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Essay Question 5

Discuss the fundamental trade-offs between value-based methods (like DQN) and policy-based methods (like REINFORCE) across the dimensions of: (a) sample efficiency, (b) stability and convergence, and (c) handling of different action spaces.

Answer

Explanation: Value-based and policy-based methods represent two distinct philosophies in reinforcement learning. The choice between them involves navigating a series of fundamental trade-offs.

(a) Sample Efficiency

- **Value-Based (More Efficient):** Value-based methods like DQN are typically *off-policy*. This is their key advantage. Being off-policy allows them to use an **experience replay buffer**, where past experiences can be stored and reused for many parameter updates. This dramatically increases data efficiency, as the agent can learn a great deal from a single interaction with the environment.

- **Policy-Based (Less Efficient):** Basic policy-based methods like REINFORCE are *on-policy*. They must learn from data generated by the current policy π_θ. As soon as the policy is updated, all previously collected data becomes "stale" and must be discarded. This requires the agent to constantly generate new trajectories, making them inherently less sample-efficient.

(b) Stability and Convergence

- **Value-Based (Less Stable):** Value-based methods, especially when combined with non-linear function approximation and bootstrapping (the "deadly triad"), can be notoriously unstable. Their learning can oscillate or diverge. This is partly because a small change in a Q-value can lead to a large, discontinuous change in the policy due to the 'argmax' operation.

- **Policy-Based (More Stable):** Policy-based methods often exhibit smoother and more stable convergence. They directly optimize the policy parameters, and a small change in θ results in a small, smooth change in the action probabilities. This makes the learning process less prone to drastic shifts and oscillations. Furthermore, theoretical work suggests that the optimal policy is often a simpler function to learn than the optimal value function, which may also contribute to their stability.

(c) Handling of Action Spaces

- **Value-Based (Suited for Discrete Actions):** Standard value-based methods like DQN are best suited for problems with **small, discrete action spaces**. The policy is derived by finding the action that maximizes the Q-values, which is a straightforward 'argmax' operation over a finite set. This maximization becomes intractable in continuous action spaces where there are infinitely many actions.

- **Policy-Based (Suited for Discrete and Continuous Actions):** Policy-based methods naturally handle **both discrete and continuous action spaces**. For continuous actions, the policy network can output the parameters of a continuous probability distribution (e.g., the mean and standard deviation of a Gaussian). An action is then sampled from this distribution. This provides a seamless and powerful mechanism for control in domains like robotics, making policy gradient methods the dominant approach for such tasks.

Essay Question 30

Explain the core architectural difference between Actor-Critic methods and their parent algorithms (pure Policy Gradient and pure Value-Based methods). How does the Critic's role in an Actor-Critic framework address the primary weakness of the REINFORCE algorithm?

Answer

Explanation: Actor-Critic methods represent a hybrid architecture that seeks to combine the best features of both pure policy-based and pure value-based methods, creating a more stable and efficient learning algorithm.

Core Architectural Difference:

- A pure **Value-Based Method** (like DQN) learns one function: a value function (e.g., $\hat{q}(s, a; w)$). The policy is implicit and derived from this value function.
- A pure **Policy Gradient Method** (like REINFORCE) learns one function: a policy ($\pi_\theta(a|s)$). The value of an action is estimated using a noisy Monte Carlo return on-the-fly but is not stored in a persistent function.
- An **Actor-Critic Method** learns **two distinct functions**:
  1. **The Actor:** This is a policy network, $\pi_\theta(a|s)$, which is responsible for controlling the agent and selecting actions. It is analogous to the network learned in policy gradient methods.
  2. **The Critic:** This is a value network, such as $\hat{v}(s; w)$ or $\hat{q}(s, a; w)$, which is responsible for evaluating the actions taken by the actor. It learns "how good" a state or state-action pair is, similar to a value-based method.

The two networks are trained in tandem. The Actor acts, the Critic critiques, and the Actor uses the Critic's feedback to improve.

Addressing the Weakness of REINFORCE: The primary weakness of the REINFORCE algorithm is the **high variance** of its policy gradient estimate. The update rule for REINFORCE is:

$$\theta \leftarrow \theta + \alpha \cdot G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)$$

The high variance comes entirely from the G_t term, which is the full Monte Carlo return from a single trajectory. This term is extremely noisy because it is the sum of many stochastic rewards and transitions.

Actor-Critic methods directly address this by replacing the high-variance Monte Carlo return with a lower-variance, more stable estimate from the Critic. One of the most common Actor-Critic update schemes uses the **TD Error** as the learning signal for the actor. The Critic first computes the TD Error:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}; w) - \hat{v}(S_t; w)$$

This TD error, while being a biased estimate of the advantage of the action, has much lower variance than the full return G_t. The Actor's parameters are then updated using this TD error instead of G_t:

$$\theta \leftarrow \theta + \alpha \cdot \delta_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)$$

By doing this, the Actor-Critic method achieves a much better trade-off:

- It keeps the benefits of a policy gradient approach from the **Actor** (stable updates, effectiveness in continuous action spaces).
- It replaces the noisy, high-variance MC return from REINFORCE with a low-variance, bootstrapped signal from the **Critic**.