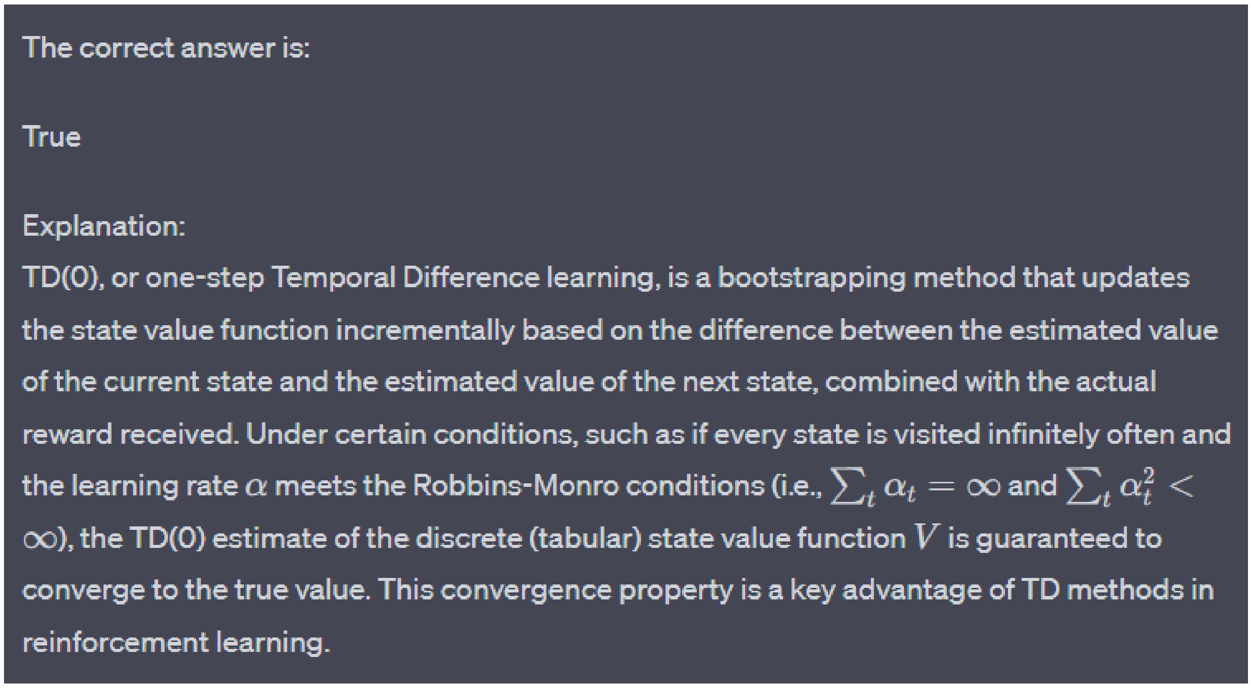
Question 1:

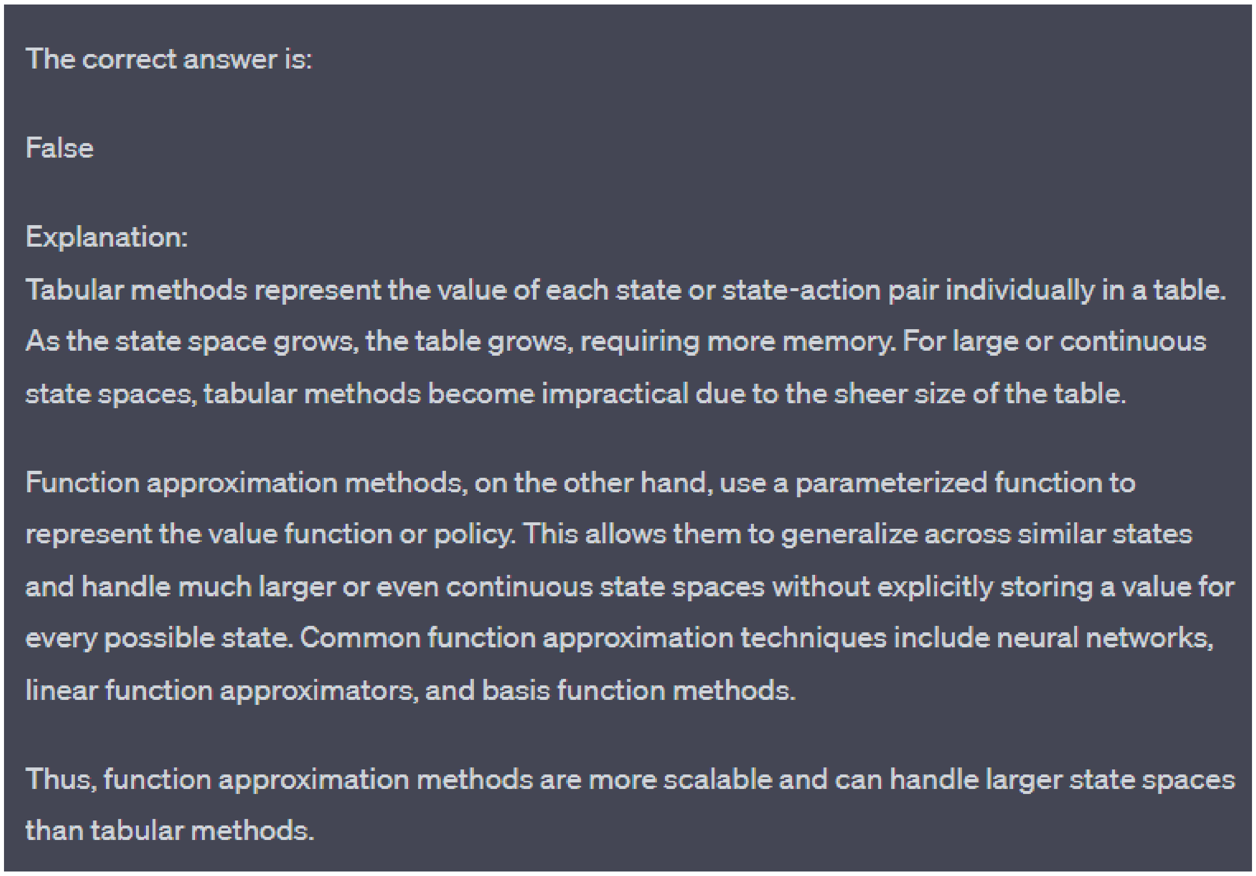


The correct answer is:

True

Explanation:
TD(0), or one-step Temporal Difference learning, is a bootstrapping method that updates the state value function incrementally based on the difference between the estimated value of the current state and the estimated value of the next state, combined with the actual reward received. Under certain conditions, such as if every state is visited infinitely often and the learning rate α meets the Robbins-Monro conditions (i.e., $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$), the TD(0) estimate of the discrete (tabular) state value function V is guaranteed to converge to the true value. This convergence property is a key advantage of TD methods in reinforcement learning.

Question 2:



The correct answer is:

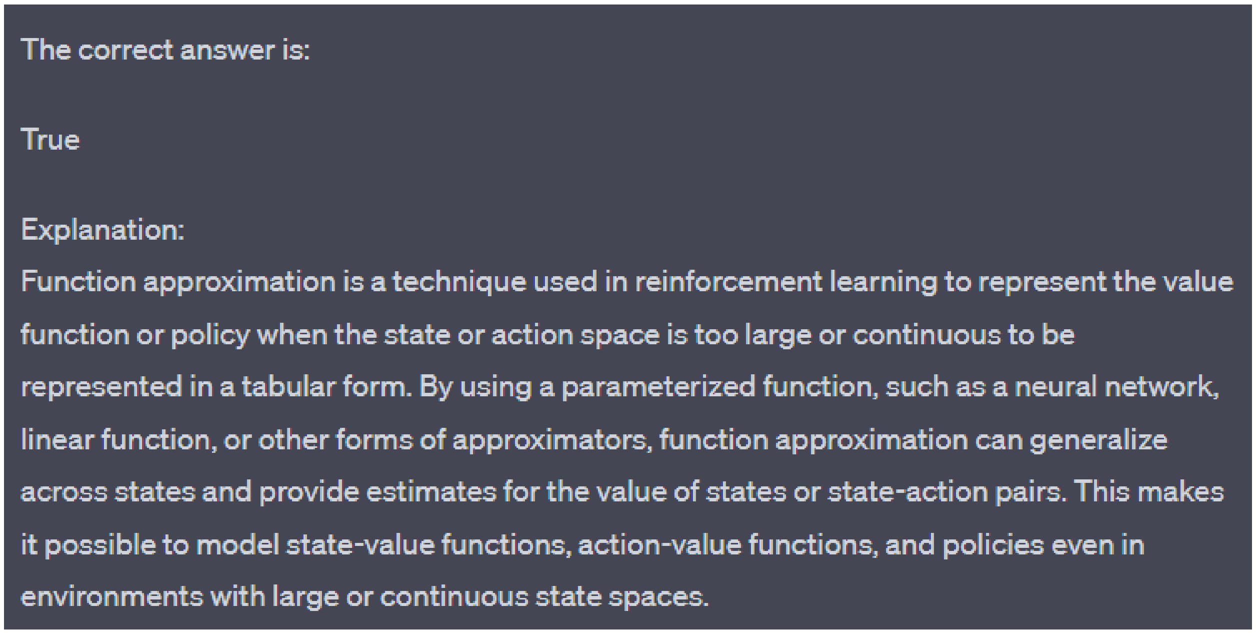
False

Explanation:
Tabular methods represent the value of each state or state-action pair individually in a table. As the state space grows, the table grows, requiring more memory. For large or continuous state spaces, tabular methods become impractical due to the sheer size of the table.

Function approximation methods, on the other hand, use a parameterized function to represent the value function or policy. This allows them to generalize across similar states and handle much larger or even continuous state spaces without explicitly storing a value for every possible state. Common function approximation techniques include neural networks, linear function approximators, and basis function methods.

Thus, function approximation methods are more scalable and can handle larger state spaces than tabular methods.

Question 3:

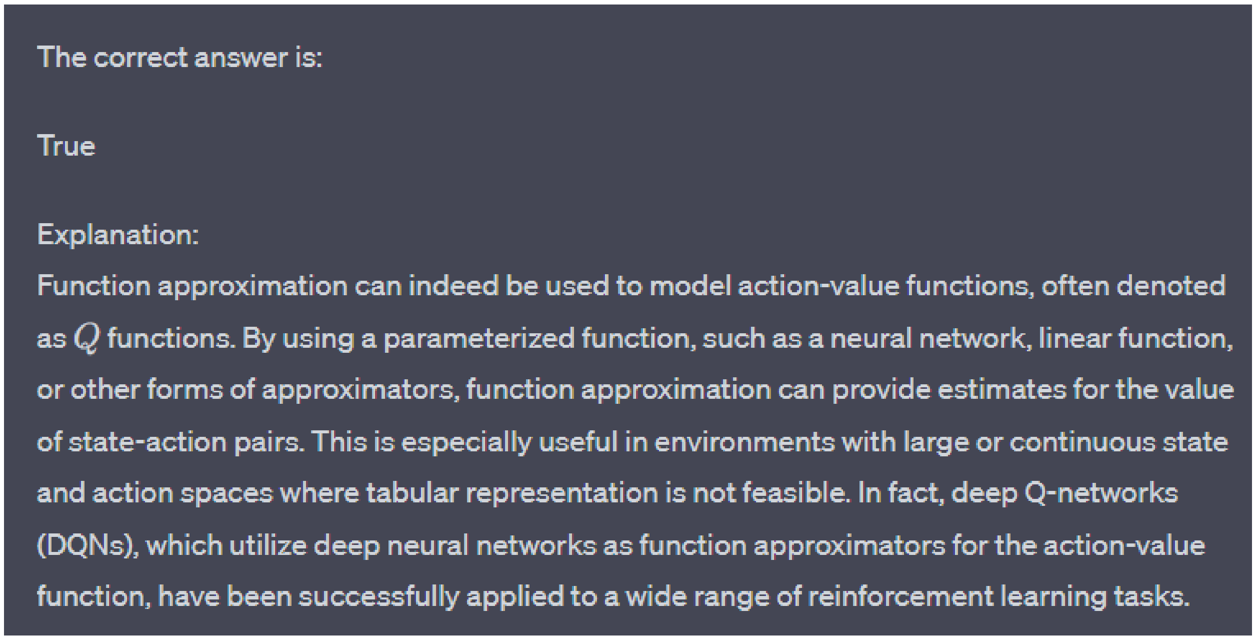


The correct answer is:

True

Explanation:
Function approximation is a technique used in reinforcement learning to represent the value function or policy when the state or action space is too large or continuous to be represented in a tabular form. By using a parameterized function, such as a neural network, linear function, or other forms of approximators, function approximation can generalize across states and provide estimates for the value of states or state-action pairs. This makes it possible to model state-value functions, action-value functions, and policies even in environments with large or continuous state spaces.

Question 4:

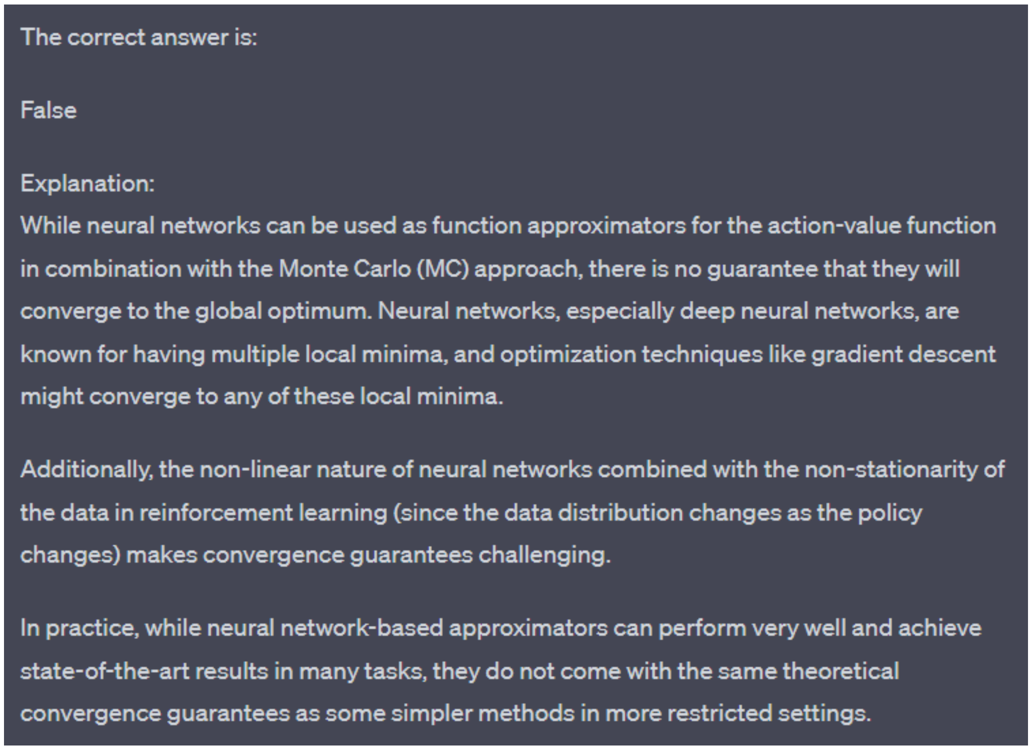


The correct answer is:

True

Explanation:
Function approximation can indeed be used to model action-value functions, often denoted as Q functions. By using a parameterized function, such as a neural network, linear function, or other forms of approximators, function approximation can provide estimates for the value of state-action pairs. This is especially useful in environments with large or continuous state and action spaces where tabular representation is not feasible. In fact, deep Q-networks (DQNs), which utilize deep neural networks as function approximators for the action-value function, have been successfully applied to a wide range of reinforcement learning tasks.

Question 5:



The correct answer is:

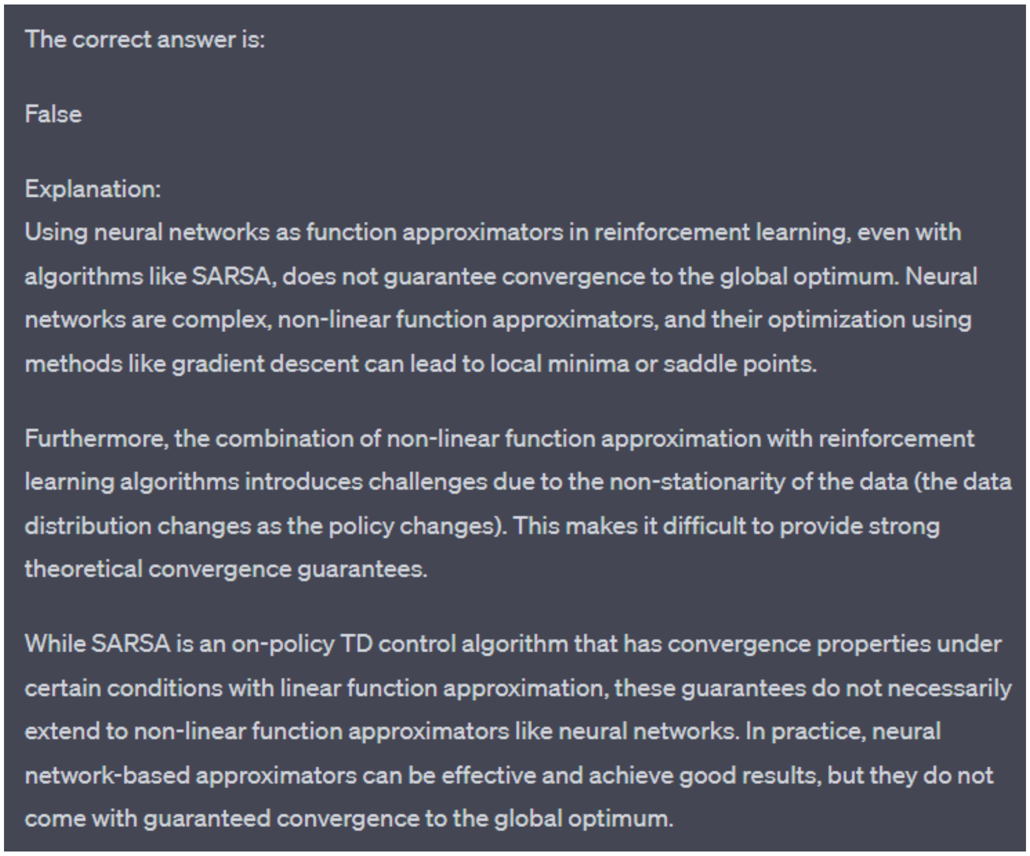
False

Explanation:
While neural networks can be used as function approximators for the action-value function in combination with the Monte Carlo (MC) approach, there is no guarantee that they will converge to the global optimum. Neural networks, especially deep neural networks, are known for having multiple local minima, and optimization techniques like gradient descent might converge to any of these local minima.

Additionally, the non-linear nature of neural networks combined with the non-stationarity of the data in reinforcement learning (since the data distribution changes as the policy changes) makes convergence guarantees challenging.

In practice, while neural network-based approximators can perform very well and achieve state-of-the-art results in many tasks, they do not come with the same theoretical convergence guarantees as some simpler methods in more restricted settings.

Question 6:



The correct answer is:

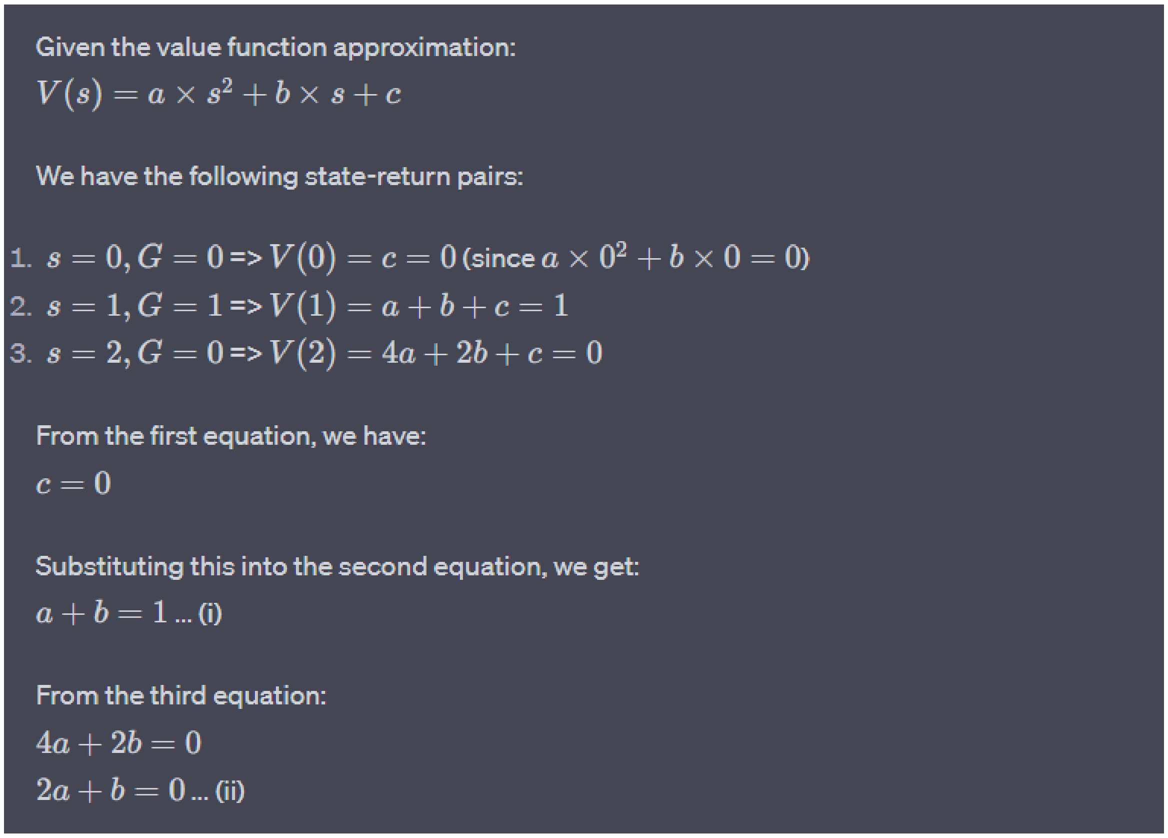
False

Explanation:
Using neural networks as function approximators in reinforcement learning, even with algorithms like SARSA, does not guarantee convergence to the global optimum. Neural networks are complex, non-linear function approximators, and their optimization using methods like gradient descent can lead to local minima or saddle points.

Furthermore, the combination of non-linear function approximation with reinforcement learning algorithms introduces challenges due to the non-stationarity of the data (the data distribution changes as the policy changes). This makes it difficult to provide strong theoretical convergence guarantees.

While SARSA is an on-policy TD control algorithm that has convergence properties under certain conditions with linear function approximation, these guarantees do not necessarily extend to non-linear function approximators like neural networks. In practice, neural network-based approximators can be effective and achieve good results, but they do not come with guaranteed convergence to the global optimum.

Question 7:



Given the value function approximation:

$V(s) = a \times s^2 + b \times s + c$

We have the following state-return pairs:

1. $s = 0, G = 0$ => $V(0) = c = 0$ (since $a \times 0^2 + b \times 0 = 0$)
2. $s = 1, G = 1$ => $V(1) = a + b + c = 1$
3. $s = 2, G = 0$ => $V(2) = 4a + 2b + c = 0$

From the first equation, we have:

$c = 0$

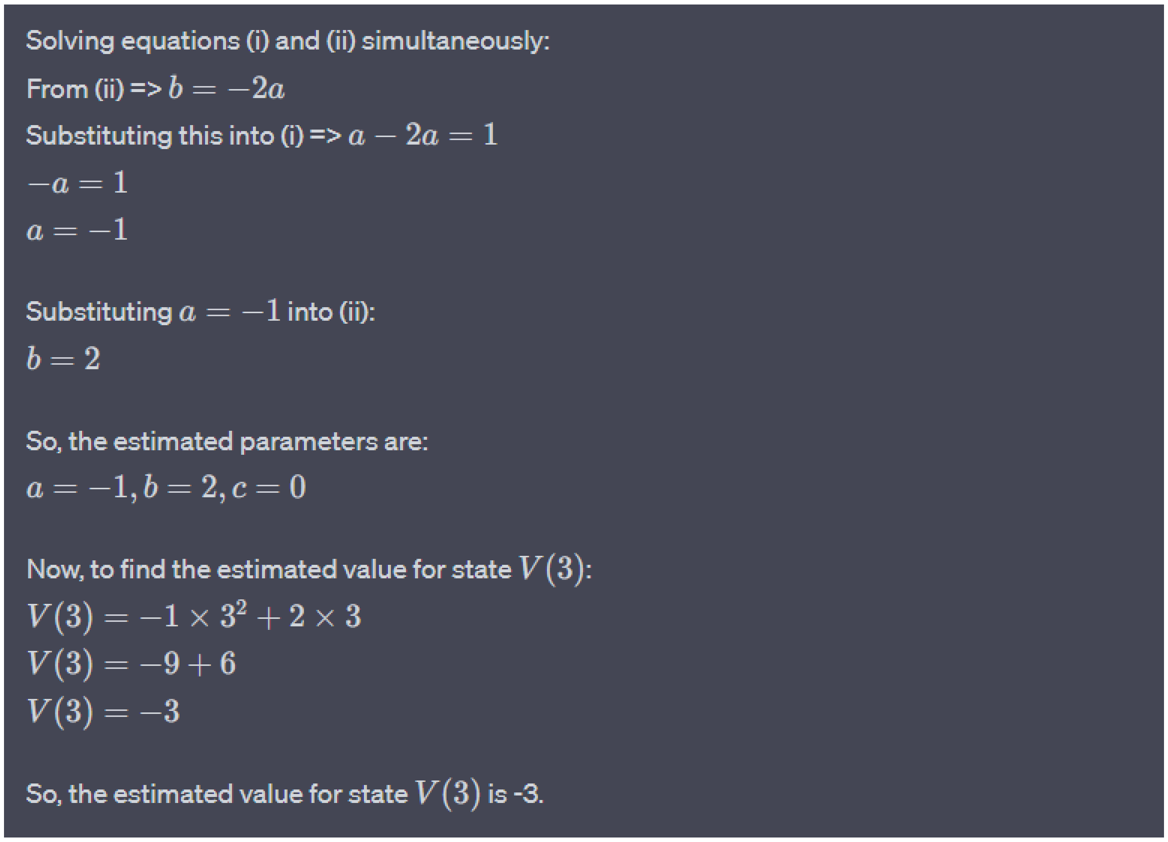
Substituting this into the second equation, we get:

$a + b = 1$ ... (i)

From the third equation:

$4a + 2b = 0$

$2a + b = 0$ ... (ii)



Solving equations (i) and (ii) simultaneously:

From (ii) => $b = -2a$

Substituting this into (i) => $a - 2a = 1$

$-a = 1$

$a = -1$

Substituting $a = -1$ into (ii):

$b = 2$

So, the estimated parameters are:

$a = -1, b = 2, c = 0$

Now, to find the estimated value for state $V(3)$:

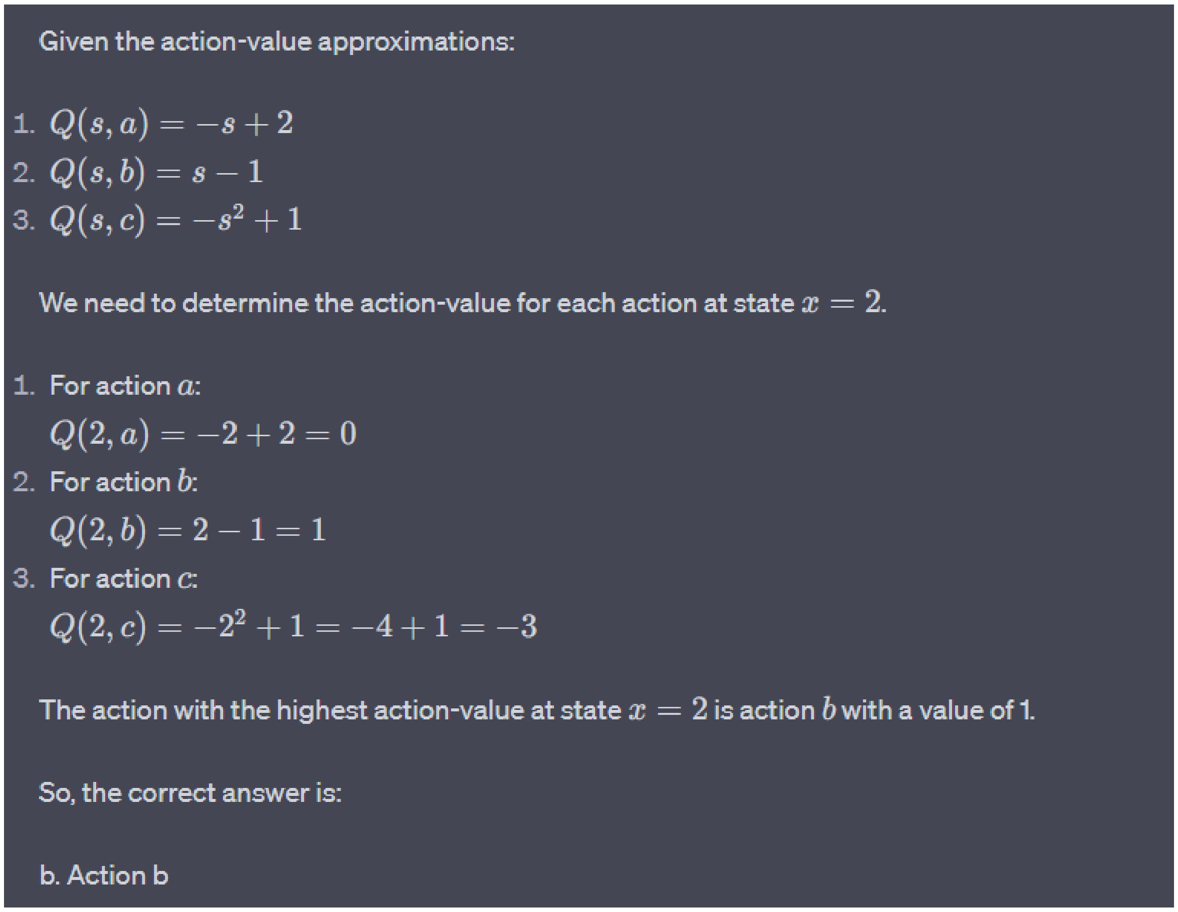
$V(3) = -1 \times 3^2 + 2 \times 3$

$V(3) = -9 + 6$

$V(3) = -3$

So, the estimated value for state $V(3)$ is -3.

Question 8:



Given the action-value approximations:

1. $Q(s,a) = -s + 2$
2. $Q(s,b) = s - 1$
3. $Q(s,c) = -s^2 + 1$

We need to determine the action-value for each action at state $x = 2$.

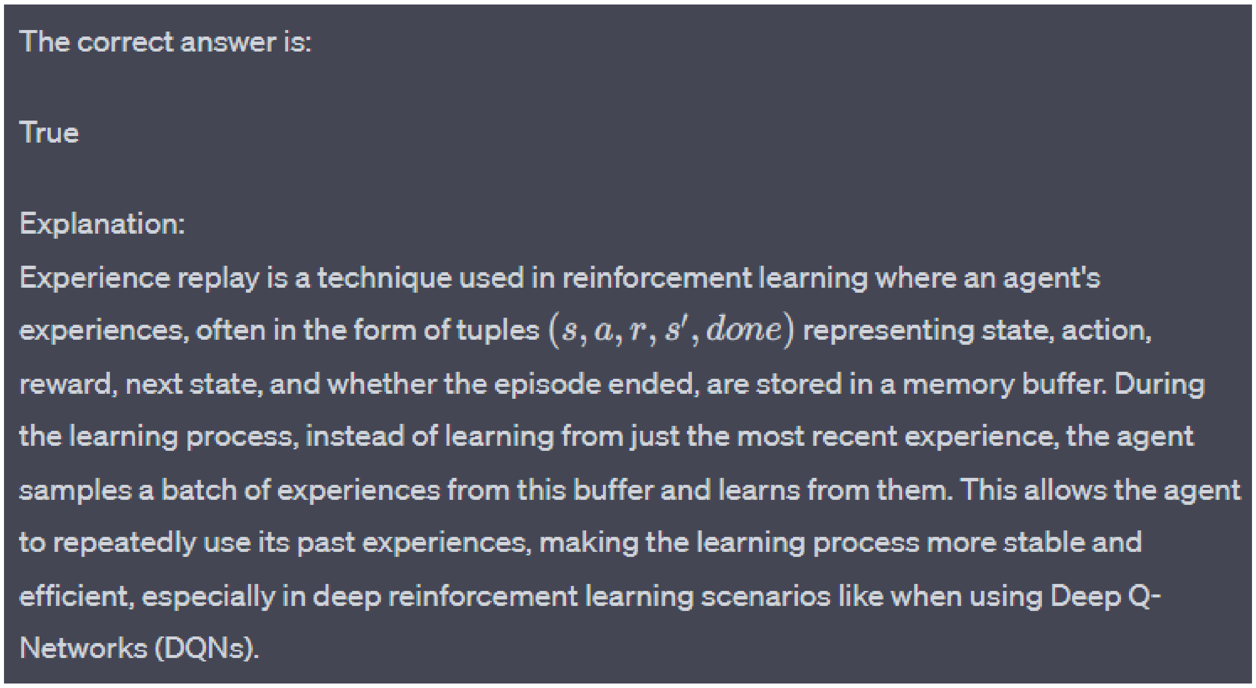
1. For action a:
   $Q(2,a) = -2 + 2 = 0$
2. For action b:
   $Q(2,b) = 2 - 1 = 1$
3. For action c:
   $Q(2,c) = -2^2 + 1 = -4 + 1 = -3$

The action with the highest action-value at state $x = 2$ is action b with a value of 1.

So, the correct answer is:

b. Action b

Question 9:

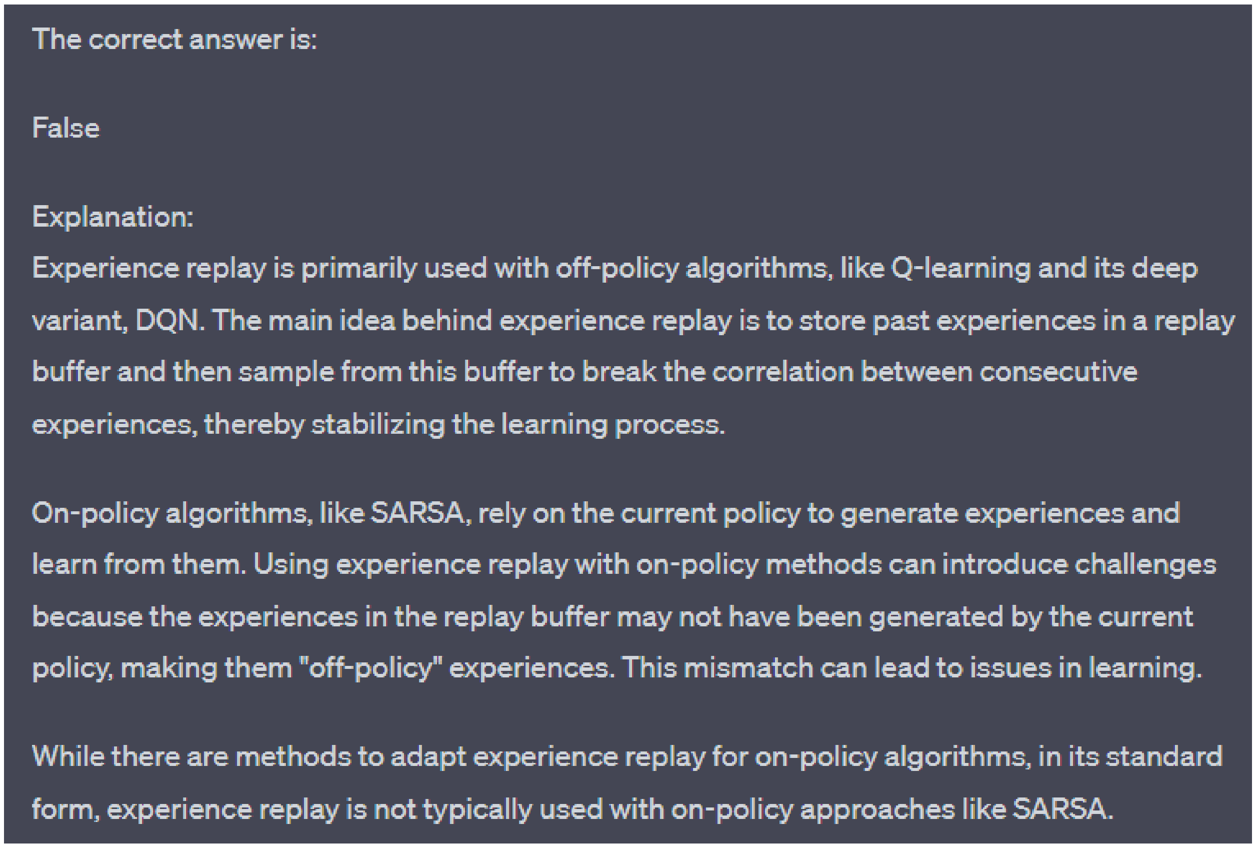


The correct answer is:

True

Explanation:
Experience replay is a technique used in reinforcement learning where an agent's experiences, often in the form of tuples $(s, a, r, s', done)$ representing state, action, reward, next state, and whether the episode ended, are stored in a memory buffer. During the learning process, instead of learning from just the most recent experience, the agent samples a batch of experiences from this buffer and learns from them. This allows the agent to repeatedly use its past experiences, making the learning process more stable and efficient, especially in deep reinforcement learning scenarios like when using Deep Q-Networks (DQNs).

Question 10:



The correct answer is:

False

Explanation:
Experience replay is primarily used with off-policy algorithms, like Q-learning and its deep variant, DQN. The main idea behind experience replay is to store past experiences in a replay buffer and then sample from this buffer to break the correlation between consecutive experiences, thereby stabilizing the learning process.

On-policy algorithms, like SARSA, rely on the current policy to generate experiences and learn from them. Using experience replay with on-policy methods can introduce challenges because the experiences in the replay buffer may not have been generated by the current policy, making them "off-policy" experiences. This mismatch can lead to issues in learning.

While there are methods to adapt experience replay for on-policy algorithms, in its standard form, experience replay is not typically used with on-policy approaches like SARSA.