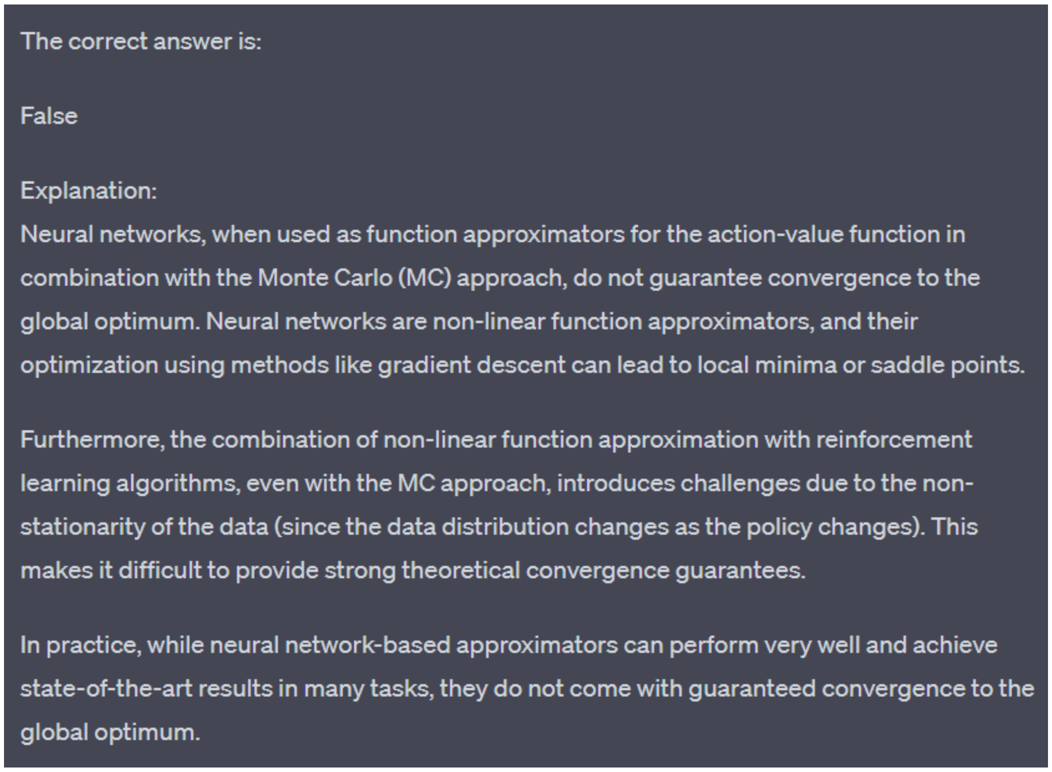
Question 1:



The correct answer is:

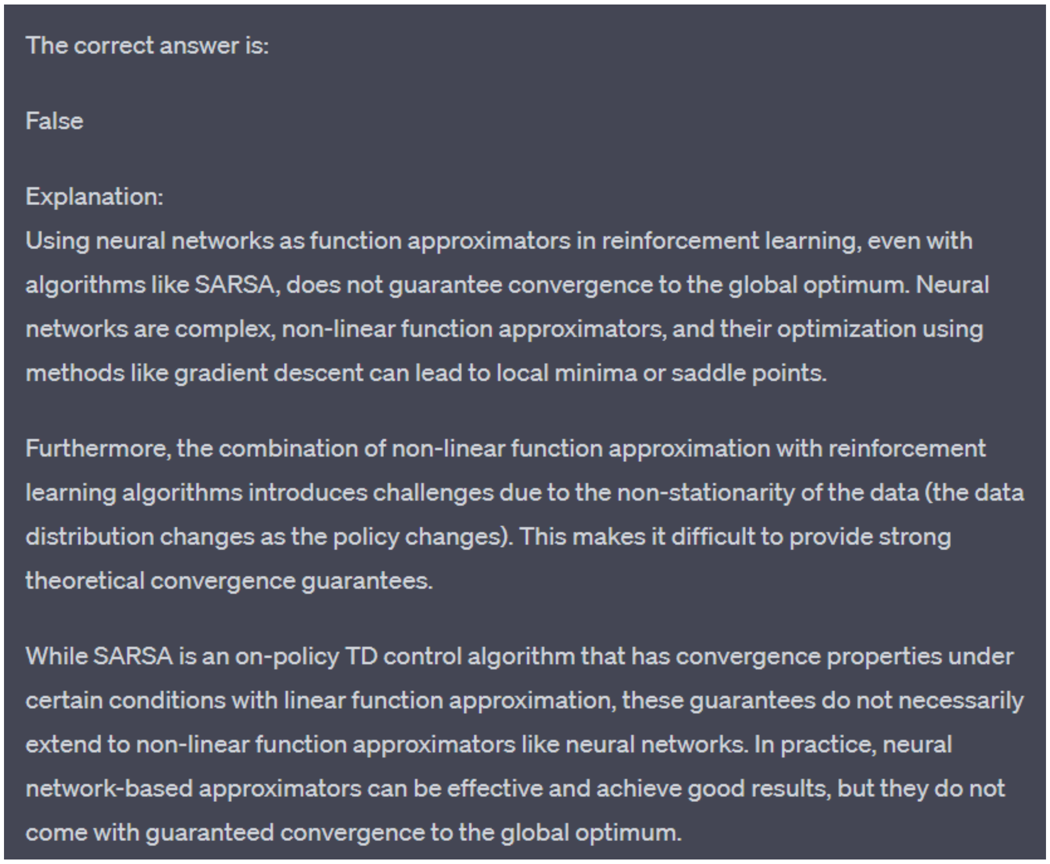
False

Explanation:
Neural networks, when used as function approximators for the action-value function in combination with the Monte Carlo (MC) approach, do not guarantee convergence to the global optimum. Neural networks are 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, even with the MC approach, introduces challenges due to the non-stationarity of the data (since the data distribution changes as the policy changes). This makes it difficult to provide strong theoretical convergence guarantees.

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 guaranteed convergence to the global optimum.

Question 2:



The correct answer is:

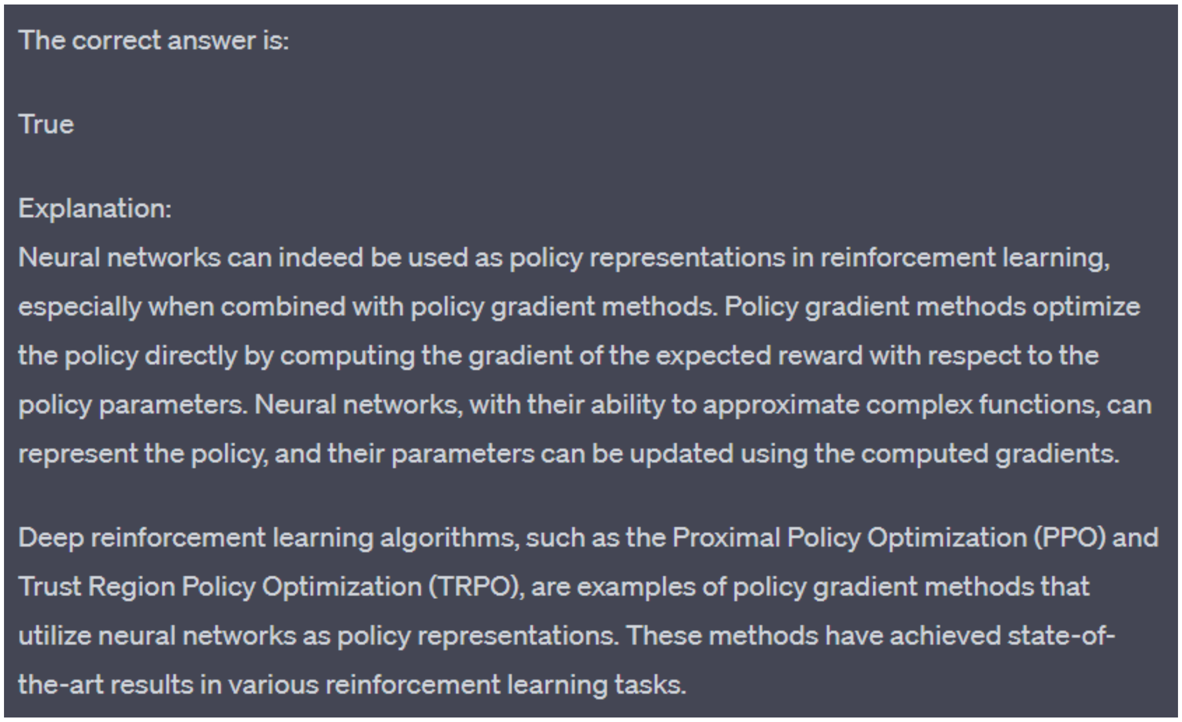
False

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 3:



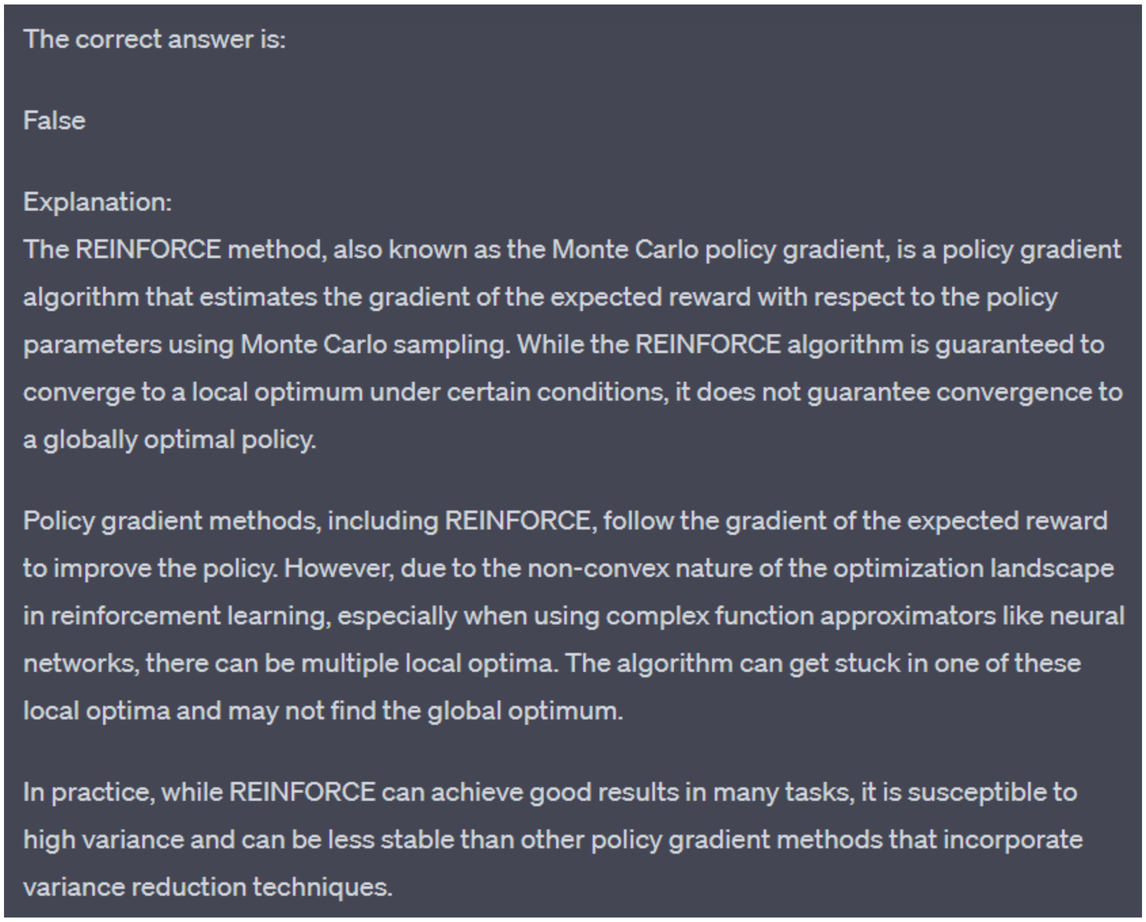
The correct answer is:

True

Explanation:
Neural networks can indeed be used as policy representations in reinforcement learning, especially when combined with policy gradient methods. Policy gradient methods optimize the policy directly by computing the gradient of the expected reward with respect to the policy parameters. Neural networks, with their ability to approximate complex functions, can represent the policy, and their parameters can be updated using the computed gradients.

Deep reinforcement learning algorithms, such as the Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), are examples of policy gradient methods that utilize neural networks as policy representations. These methods have achieved state-of-the-art results in various reinforcement learning tasks.

Question 4:



The correct answer is:

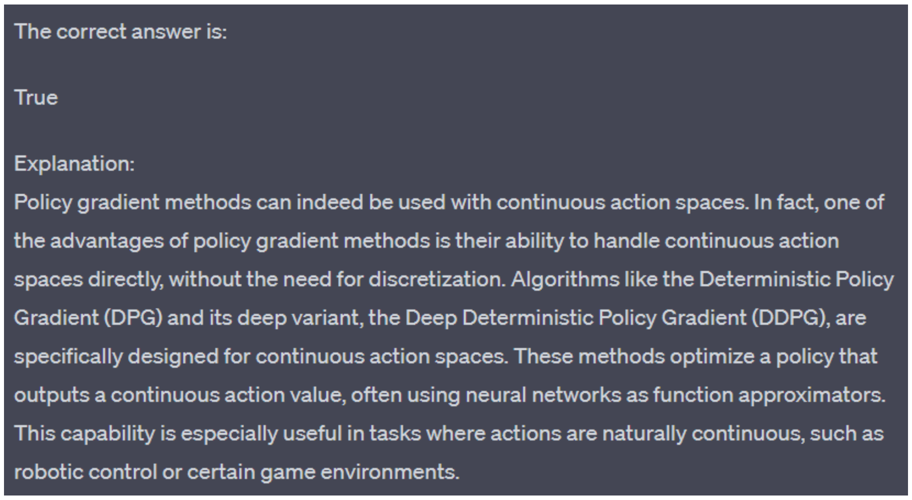
False

Explanation:
The REINFORCE method, also known as the Monte Carlo policy gradient, is a policy gradient algorithm that estimates the gradient of the expected reward with respect to the policy parameters using Monte Carlo sampling. While the REINFORCE algorithm is guaranteed to converge to a local optimum under certain conditions, it does not guarantee convergence to a globally optimal policy.

Policy gradient methods, including REINFORCE, follow the gradient of the expected reward to improve the policy. However, due to the non-convex nature of the optimization landscape in reinforcement learning, especially when using complex function approximators like neural networks, there can be multiple local optima. The algorithm can get stuck in one of these local optima and may not find the global optimum.

In practice, while REINFORCE can achieve good results in many tasks, it is susceptible to high variance and can be less stable than other policy gradient methods that incorporate variance reduction techniques.

Question 5:

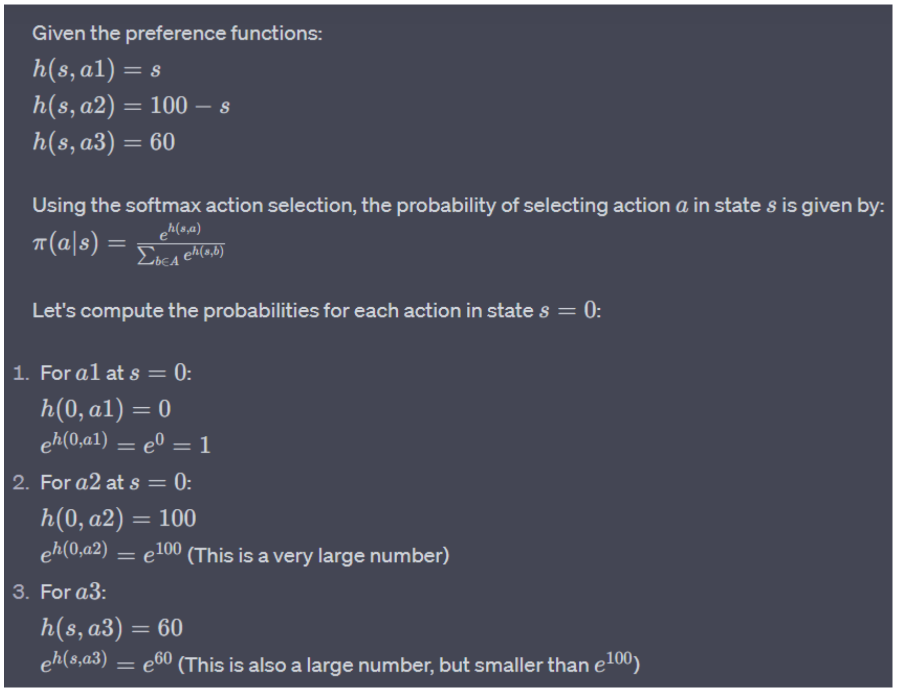


The correct answer is:

True

Explanation:
Policy gradient methods can indeed be used with continuous action spaces. In fact, one of the advantages of policy gradient methods is their ability to handle continuous action spaces directly, without the need for discretization. Algorithms like the Deterministic Policy Gradient (DPG) and its deep variant, the Deep Deterministic Policy Gradient (DDPG), are specifically designed for continuous action spaces. These methods optimize a policy that outputs a continuous action value, often using neural networks as function approximators. This capability is especially useful in tasks where actions are naturally continuous, such as robotic control or certain game environments.

Question 6:

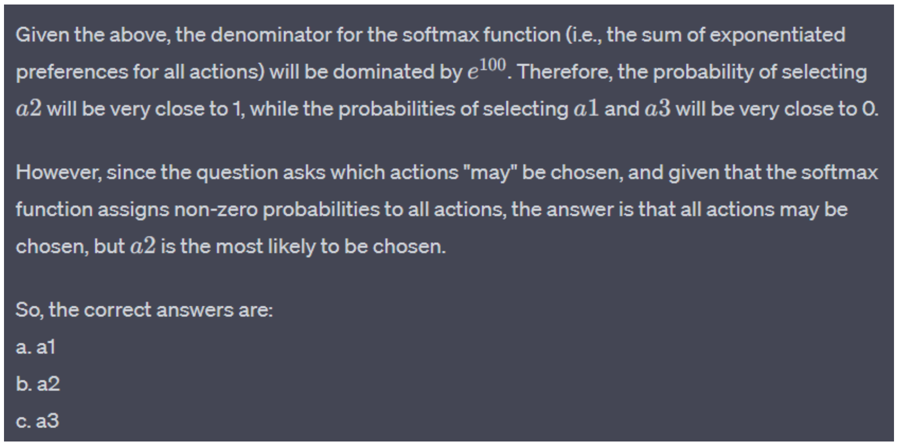


Given the preference functions:
$h(s, a1) = s$
$h(s, a2) = 100 - s$
$h(s, a3) = 60$

Using the softmax action selection, the probability of selecting action a in state s is given by:
$\pi(a|s) = \frac{e^{h(s,a)}}{\sum_{b \in A} e^{h(s,b)}}$

Let's compute the probabilities for each action in state $s = 0$:

1. For $a1$ at $s = 0$:
   $h(0, a1) = 0$
   $e^{h(0,a1)} = e^0 = 1$
2. For $a2$ at $s = 0$:
   $h(0, a2) = 100$
   $e^{h(0,a2)} = e^{100}$ (This is a very large number)
3. For $a3$:
   $h(s, a3) = 60$
   $e^{h(s,a3)} = e^{60}$ (This is also a large number, but smaller than e^{100})

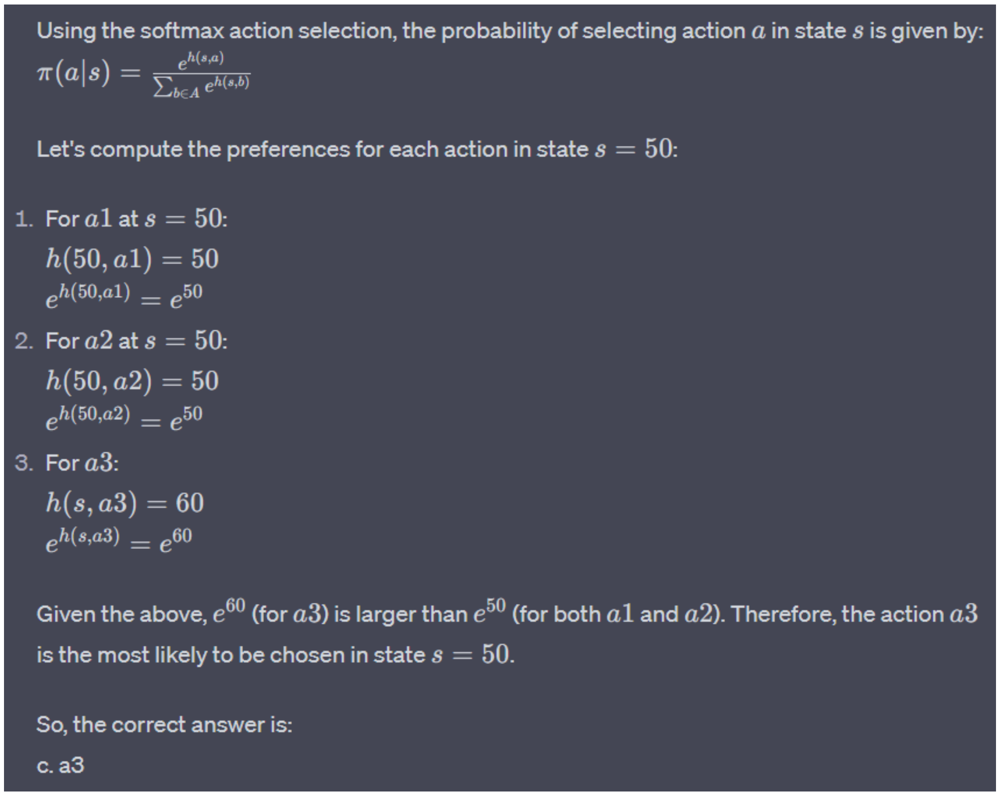


Given the above, the denominator for the softmax function (i.e., the sum of exponentiated preferences for all actions) will be dominated by e^{100}. Therefore, the probability of selecting $a2$ will be very close to 1, while the probabilities of selecting $a1$ and $a3$ will be very close to 0.

However, since the question asks which actions "may" be chosen, and given that the softmax function assigns non-zero probabilities to all actions, the answer is that all actions may be chosen, but $a2$ is the most likely to be chosen.

So, the correct answers are:
a. a1
b. a2
c. a3

Question 7:



Using the softmax action selection, the probability of selecting action a in state s is given by:
$\pi(a|s) = \frac{e^{h(s,a)}}{\sum_{b\in A} e^{h(s,b)}}$

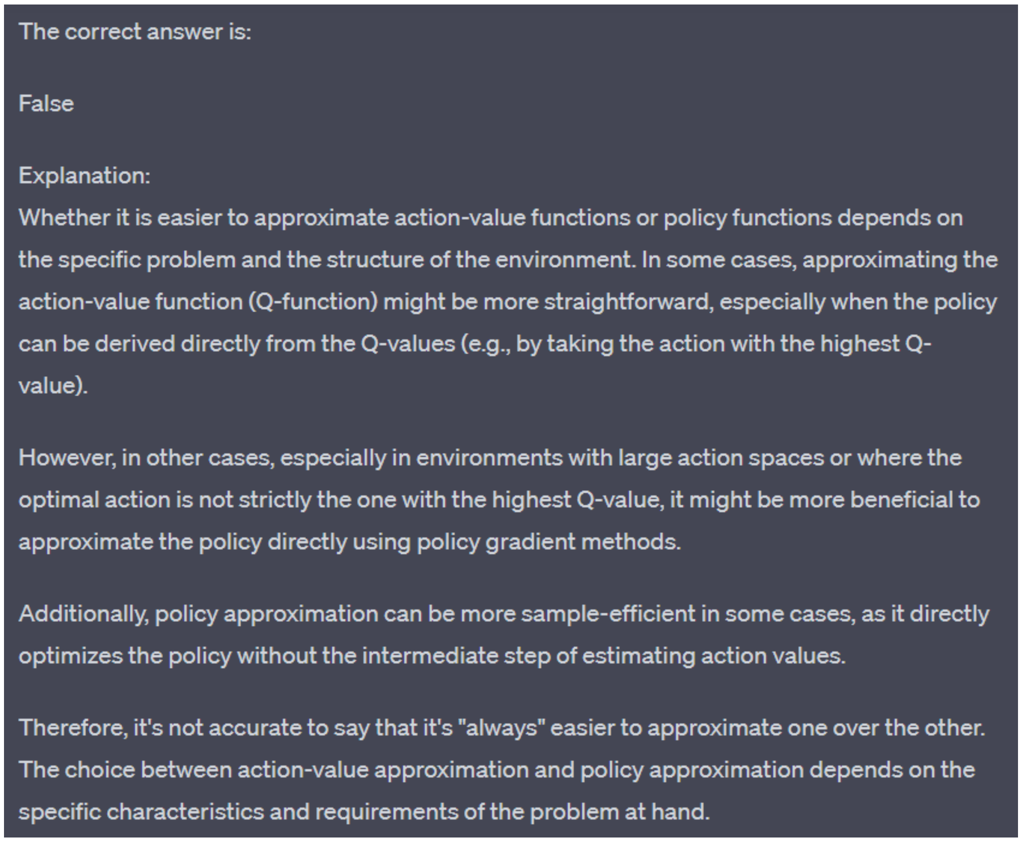
Let's compute the preferences for each action in state $s = 50$:

1. For $a1$ at $s = 50$:
   $h(50, a1) = 50$
   $e^{h(50,a1)} = e^{50}$
2. For $a2$ at $s = 50$:
   $h(50, a2) = 50$
   $e^{h(50,a2)} = e^{50}$
3. For $a3$:
   $h(s, a3) = 60$
   $e^{h(s,a3)} = e^{60}$

Given the above, e^{60} (for $a3$) is larger than e^{50} (for both $a1$ and $a2$). Therefore, the action $a3$ is the most likely to be chosen in state $s = 50$.

So, the correct answer is:
c. a3

Question 8:



The correct answer is:

False

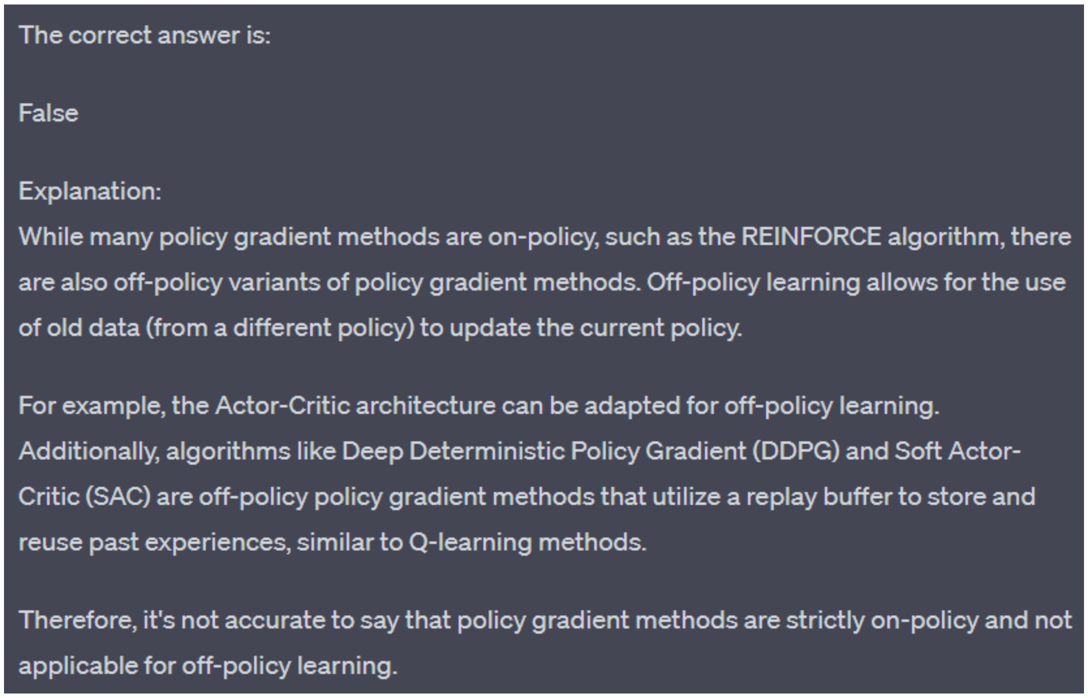
Explanation:
Whether it is easier to approximate action-value functions or policy functions depends on the specific problem and the structure of the environment. In some cases, approximating the action-value function (Q-function) might be more straightforward, especially when the policy can be derived directly from the Q-values (e.g., by taking the action with the highest Q-value).

However, in other cases, especially in environments with large action spaces or where the optimal action is not strictly the one with the highest Q-value, it might be more beneficial to approximate the policy directly using policy gradient methods.

Additionally, policy approximation can be more sample-efficient in some cases, as it directly optimizes the policy without the intermediate step of estimating action values.

Therefore, it's not accurate to say that it's "always" easier to approximate one over the other. The choice between action-value approximation and policy approximation depends on the specific characteristics and requirements of the problem at hand.

Question 9:



The correct answer is:

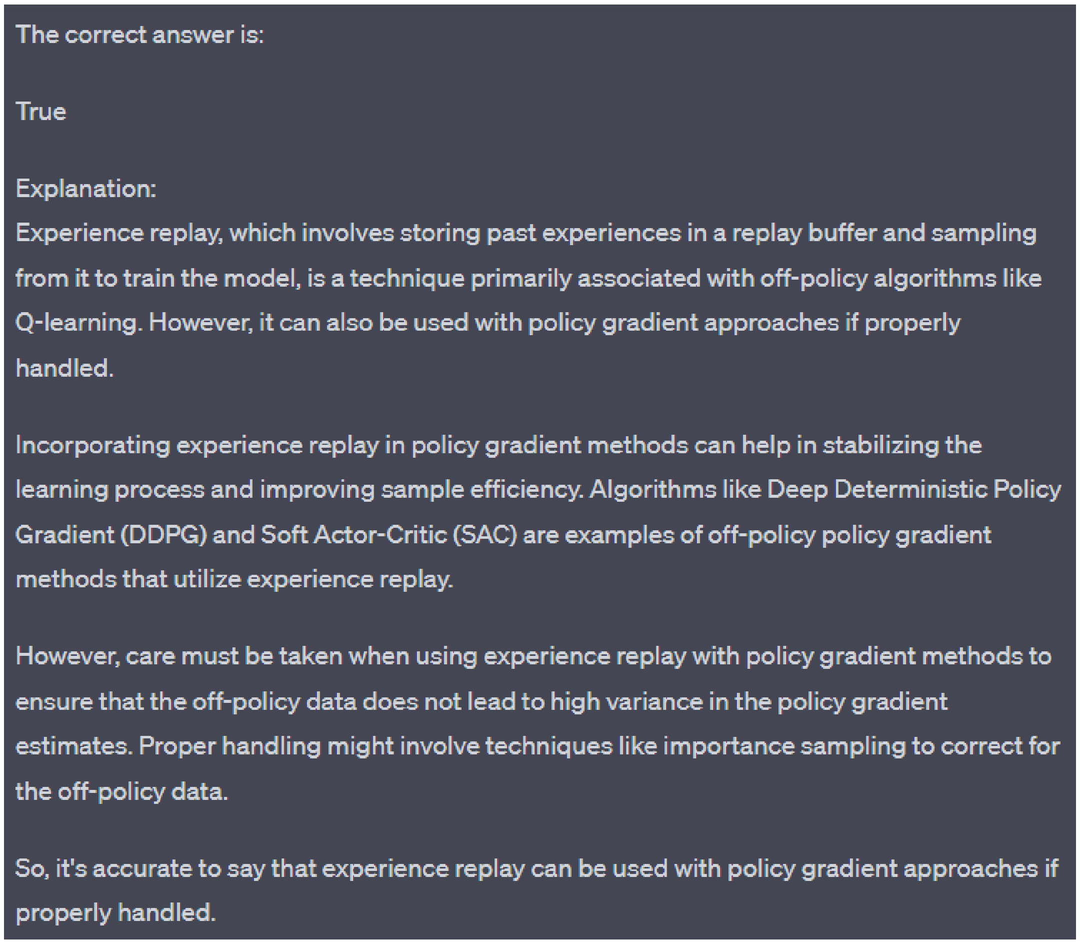
False

Explanation:
While many policy gradient methods are on-policy, such as the REINFORCE algorithm, there are also off-policy variants of policy gradient methods. Off-policy learning allows for the use of old data (from a different policy) to update the current policy.

For example, the Actor-Critic architecture can be adapted for off-policy learning. Additionally, algorithms like Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC) are off-policy policy gradient methods that utilize a replay buffer to store and reuse past experiences, similar to Q-learning methods.

Therefore, it's not accurate to say that policy gradient methods are strictly on-policy and not applicable for off-policy learning.

Question 10:



The correct answer is:

True

Explanation:
Experience replay, which involves storing past experiences in a replay buffer and sampling from it to train the model, is a technique primarily associated with off-policy algorithms like Q-learning. However, it can also be used with policy gradient approaches if properly handled.

Incorporating experience replay in policy gradient methods can help in stabilizing the learning process and improving sample efficiency. Algorithms like Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC) are examples of off-policy policy gradient methods that utilize experience replay.

However, care must be taken when using experience replay with policy gradient methods to ensure that the off-policy data does not lead to high variance in the policy gradient estimates. Proper handling might involve techniques like importance sampling to correct for the off-policy data.

So, it's accurate to say that experience replay can be used with policy gradient approaches if properly handled.