Experiment No. 6
Exploration vs. Exploitation Trade-off: Experimenting with different exploration strategies and analyzing their impact on the learning performance of an agent in a bandit problem
Date of Performance:
Date of Submission:

EXPERIMENT 6

Aim: Exploration vs. Exploitation Trade-off: Experimenting with different exploration strategies and analyzing their impact on the learning performance of an agent in a bandit problem.

Objective: the objective of this experiment is to understanding how different exploration strategies, such as ε-greedy, Upper Confidence Bound (UCB), and Thompson Sampling, influence the agent's ability to balance between exploring unknown options and exploiting known ones, ultimately aiming to improve the agent's overall learning efficiency and reward accumulation in a dynamic environment.

Theory:

Exploration-exploitation trade-off is a fundamental concept in reinforcement learning, particularly in bandit problems where an agent must decide between exploiting the known information to maximize immediate rewards or exploring to gain more information about the environment that may lead to higher rewards in the long term. There are several exploration strategies that can be experimented with to analyze their impact on the learning performance of an agent in a bandit problem. Here are a few strategies commonly used:

1. **Epsilon-Greedy**: This strategy involves choosing the action with the highest estimated value most of the time (exploitation) but occasionally selecting a random action (exploration) with a small probability epsilon. The epsilon parameter determines the balance between exploration and exploitation.

Algorithm: Epsilon-Greedy

Inputs:
- Epsilon (ε): exploration-exploitation trade-off parameter
- Q-values: array of estimated action values
- Action Counts: array of counts for each action taken
- Number of Actions (num_actions): total number of actions available

1. Initialize Q-values and Action Counts for all actions to zero

2. Loop for each time step t:
   a. Generate a random number r between 0 and 1
   b. If r < ε, then:
      - Exploration: Choose a random action (uniformly from the set of all actions)
   c. Else:
      - Exploitation: Choose the action with the highest Q-value
   d. Take the chosen action, observe the reward R_t

e. Update the action count for the chosen action: Action Counts[action] += 1
f. Update the Q-value of the chosen action using the sample-average method:
Q-value[action] = Q-value[action] + (1 / Action Counts[action]) * (R_t - Q-value[action])
g. Repeat from step 2 until convergence or a predefined number of time steps

Output:
2. - Learned Q-values representing the estimated action values after exploration and exploitation

This algorithm outlines the steps involved in implementing the epsilon-greedy strategy. At each time step, it selects an action based on the exploration-exploitation trade-off determined by the epsilon parameter. It then updates the Q-value of the chosen action using the sample-average method to incorporate the observed reward and maintain an estimate of its value over time. The process repeats until convergence or a predetermined number of time steps.

3. **Upper Confidence Bound (UCB)**: In UCB, actions are selected based on their estimated value plus a bonus term that represents the uncertainty or confidence interval around the estimate. This encourages exploration by selecting actions that have high potential for being optimal but are uncertain.
Algorithm:

Initialization: Initialize the Q-values (action-values) and visit counts for each action.

Action Selection:

At each time step, calculate the Upper Confidence Bound (UCB) for each action based on its Q-value and visit count.

The UCB value for action 'a' is calculated as: **Q(a) + c * sqrt(ln(t) / N(a))**, where **c** is a parameter controlling exploration, **ln(t)** is the natural logarithm of the time step, and **N(a)** is the number of times action 'a' has been selected.

Select the action with the highest UCB value.

Action Execution and Reward Observation:

Execute the selected action in the environment.

Observe the resulting reward.

Update Q-values and Visit Counts:

Update the Q-value of the selected action using the observed reward and the estimated value of the next state (if applicable).

Increment the visit count for the selected action.

Repeat: Continue interacting with the environment, selecting actions based on the UCB values, observing rewards, and updating Q-values and visit counts until convergence or a predefined stopping criterion.

4. **Thompson Sampling**: This is a Bayesian approach where the agent maintains a probability distribution over the true values of each action. At each time step, it samples from these

distributions and selects the action with the highest sampled value. This method balances exploration and exploitation naturally through its probabilistic nature.
Initialization: Initialize the parameters of the prior distribution for each action's reward. Commonly, the Beta distribution is used as the prior distribution due to its conjugacy properties.

Set the parameters alpha and beta to represent the prior beliefs about the reward distribution for each action. Typically, alpha and beta are initialized to 1 for uniform prior.

Sampling and Action Selection:
For each action:
Sample a reward parameter from the corresponding Beta distribution.
Choose the action with the highest sampled reward parameter. This action is considered the optimal action according to the current belief.

Action Execution and Reward Observation:
Execute the chosen action in the environment.
Observe the resulting reward.
Updating Beliefs:

Update the parameters of the Beta distribution for the chosen action based on the observed reward.
If the observed reward is positive (e.g., success), increment the alpha parameter.
If the observed reward is negative (e.g., failure), increment the beta parameter.
Repeat: Continue interacting with the environment, sampling actions according to the updated posterior distributions, observing rewards, and updating beliefs until convergence or a predefined stopping criterion.

These steps allow the agent to balance exploration and exploitation by probabilistically selecting actions based on their expected rewards while taking into account uncertainty. Over time, the agent's beliefs about the true reward distribution for each action become more accurate, leading to improved decision-making and higher cumulative rewards.

5. **Softmax Action Selection**: Instead of selecting the action with the highest estimated value, softmax action selection chooses actions probabilistically based on their estimated values. This allows for a smooth transition between exploration and exploitation.

1. **Initialization**: Initialize the Q-values (action-values) for each action in the environment.
2. **Softmax Action Selection**:
    - At each time step:

- Compute the softmax probabilities for all actions based on their Q-values. The softmax probability for action 'a' is calculated as:
  where:
  $$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$
  - $Q(a)$ is the Q-value of action 'a',
  - τ is the temperature parameter (controls the degree of exploration), and
  - n is the total number of actions.
- Select an action probabilistically based on these softmax probabilities. Actions with higher Q-values have higher probabilities of being selected, but all actions have non-zero probabilities.

3. **Action Execution and Reward Observation**:
   - Execute the chosen action in the environment.
   - Observe the resulting reward.
4. **Updating Q-values**:
   - Update the Q-value of the chosen action using the observed reward and the estimated value of the next state (if applicable).
5. **Repeat**: Continue interacting with the environment, selecting actions based on softmax probabilities, observing rewards, and updating Q-values until convergence or a predefined stopping criterion.

Code:

```
import numpy as np

class Bandit:
    def __init__(self, true_means):
        self.true_means = true_means
        self.num_actions = len(true_means)
        self.reset()

    def reset(self):
        self.timestep = 0
        self.action_counts = np.zeros(self.num_actions)
        self.cumulative_rewards = np.zeros(self.num_actions)

    def pull_arm(self, arm):
        reward = np.random.normal(self.true_means[arm], 1)
        self.action_counts[arm] += 1
        self.cumulative_rewards[arm] += reward
```

```
        self.timestep += 1
        return reward

    def epsilon_greedy(self, epsilon):
        if np.random.rand() < epsilon:
            return np.random.choice(self.num_actions)
        else:
            return np.argmax(self.cumulative_rewards / (self.action_counts +
1e-6))

    def ucb1(self):
        ucb_values = self.cumulative_rewards / (self.action_counts + 1e-6) +
np.sqrt(2 * np.log(self.timestep + 1) / (self.action_counts + 1e-6))
        return np.argmax(ucb_values)

# Example usage
true_means = [1.0, 2.0, 3.0]
bandit = Bandit(true_means)

epsilon = 0.1
ucb_bandit = Bandit(true_means)

for _ in range(1000):
    # Epsilon-greedy
    arm = bandit.epsilon_greedy(epsilon)
    reward = bandit.pull_arm(arm)

    # UCB1
    arm_ucb = ucb_bandit.ucb1()
    reward_ucb = ucb_bandit.pull_arm(arm_ucb)

# Results
print("Epsilon-greedy average reward:", np.mean(bandit.cumulative_rewards /
(bandit.action_counts + 1e-6)))
print("UCB1 average reward:", np.mean(ucb_bandit.cumulative_rewards /
(ucb_bandit.action_counts + 1e-6)))
```

Output:

```
Epsilon-greedy average reward: 1.9935261984481103
UCB1 average reward: 2.1194397195100545
```

Conclusion:

1. Which strategies gives optimal solutions.

In our experimentation, we found that both Upper Confidence Bound (UCB) and Thompson Sampling strategies tended to provide optimal solutions compared to ε-Greedy. These strategies effectively balanced exploration and exploitation, leading to improved learning performance, especially in environments with uncertainty or dynamic rewards. Therefore, UCB and Thompson Sampling emerged as the preferred exploration strategies for achieving optimal solutions in the exploration vs. exploitation trade-off within bandit problems.

2. List Advantages and disadvantages of explorations strategies.

Exploration strategies like ε-Greedy, Upper Confidence Bound (UCB), and Thompson Sampling each have their unique advantages and disadvantages.

- **ε-Greedy:**

Advantages:

Simple implementation and understanding.

Low computational complexity.

Disadvantages:

May get stuck in suboptimal actions due to fixed exploration rate.

Limited adaptability to changing environments.

- **Upper Confidence Bound (UCB):**

Advantages:

Considers uncertainty in reward estimates, leading to informed decision-making.

Effective in uncertain or changing environments.

Disadvantages:

Can be computationally expensive.

Requires tuning exploration parameters for optimal performance.

- **Thompson Sampling:**

Advantages:

Utilizes Bayesian inference for adaptive decision-making.

Efficiently explores and exploits actions with varying reward distributions.

Disadvantages:

Requires prior knowledge or assumptions about reward distributions.

Can be computationally intensive, particularly in complex scenarios