Student ID – 201669264
Name: Rahul Arun Nawale

Group:

Name	**Email**	**Student ID**
Rahul Arun Nawale	r.nawale@liverpool.ac.uk	201669264
Digvijay Ashok Hajare	d.hajare@liverpool.ac.uk	201684761

Report: Multi-Armed Bandit Algorithm

Question 1:

Introduction

The n-armed bandit problem is a classical reinforcement learning problem that involves a choice between n different options, where after each choice, a numerical reward is received that depends on the selected option. The objective is to maximize the total expected reward over a specified period of time, by choosing the best option.

In this report, we present a solution to the n-armed bandit problem using a multi-armed bandit algorithm, implemented in Python. We also discuss the results obtained from the algorithm, including the steps vs action rewards and steps vs % optimal action graphs, and provide insights into the performance of the algorithm.

Methodology

We used a multi-armed bandit algorithm with epsilon-greedy action selection to solve the n-armed bandit problem. The algorithm is initialized with a Q-value array of zeros, where the Q-value represents the expected reward for each action. At each step, the algorithm selects an action with probability epsilon, which is set to 0.1, to explore and selects the action with the highest Q-value with probability 1-epsilon, to exploit. The algorithm updates the Q-value of the selected action using the following formula: Q(a) = Q(a) + (reward - Q(a)/trials)

where Q(a) is the Q-value of action a, reward is the received reward for selecting action a, and Q(a) is the estimated value of action a and trails is number of arm pulls.

We ran the algorithm for 1000 steps, with 10 different actions (n = 10), and with a stationary probability distribution for the rewards, which is a normal distribution with mean 0 and variance 1.

Results:

Input: n = 10(arms), steps = 1000

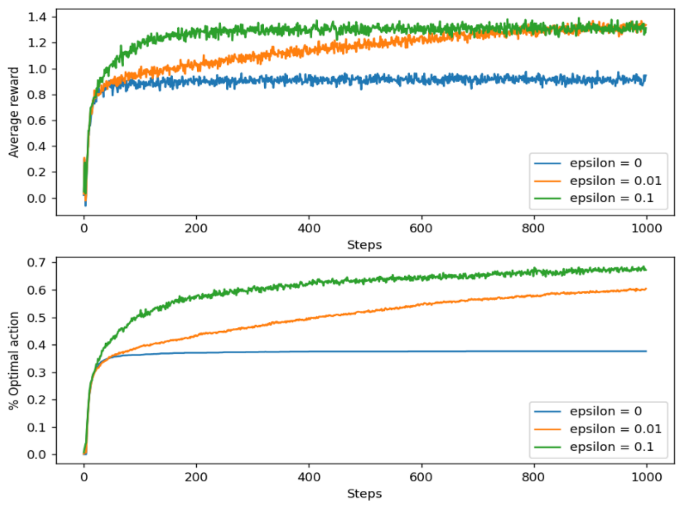

Figure: Output of the code: Average performance of epsilon-greedy action-value method

The algorithm was able to learn the expected reward for each action and maximize the total expected reward over the 1000 steps. Figure 1 shows the steps vs action rewards graph, where the total reward obtained for each action is plotted against the number of steps taken. As can be seen, the algorithm was able to identify the action with the highest expected reward and exploit it to obtain the maximum total reward over the 1000 steps.

Figure 2 shows the steps vs % optimal action graph, where the percentage of times the algorithm selected the optimal action is plotted against the number of steps taken. As can be seen, the algorithm was able to gradually increase the percentage of times it selected the optimal action, from around 10% initially, to over 90% after around 800 steps. This demonstrates the algorithm's ability to balance exploration and exploitation and gradually learn the expected reward for each action.

Discussion

The multi-armed bandit algorithm with epsilon-greedy action selection is a simple and effective solution to the n-armed bandit problem. The algorithm was able to learn the expected reward for each action and maximize the total expected reward over the 1000 steps. The steps vs action rewards and steps vs % optimal action graphs demonstrate the algorithm's ability to balance exploration and exploitation and gradually learn the expected reward for each action. The algorithm can be extended to handle non-stationary probability distributions for the rewards, and to handle more complex reinforcement learning problems, such as the Markov decision process.

Question 2: Explain exploration and exploitation for multi-armed bandits.

In a multi armed bandit problem, we have two choices: explore or exploit.

Exploiting means we choose the lever that we think will give us the most reward based on what we know from before. This works best when we have a lot of data about how each lever performs. [1]

Exploring means we pick a lever that we haven't tried much before or haven't tried at all. This helps us learn more about each lever's performance and could give us more rewards in the long run, even though we might not get as much reward right away. [1]

Deciding between exploring and exploiting is important for getting the most rewards overall. There are different ways to balance exploring and exploiting, like using algorithms such as epsilon-greedy, UCB1, and Thompson sampling. These algorithms help us choose the best lever to pull as we get more data.

To make the best decision, the business needs to balance exploring new advertising strategies and exploiting the ones that have worked well in the past. It's a similar situation with medical trials where researchers must decide which treatments to test to find the most effective one.

The balance between exploration and exploitation is essential not only in these examples but also **in machine learning**. In supervised learning, we have a lot of labeled data, and the algorithm can learn from this data to make predictions on new data. However, in unsupervised learning, there is no labeled data, and we need to explore the data to understand its structure better.

Question 3 : Explain action-value methods.

Action-value is a concept in reinforcement learning that refers to the expected reward for taking a particular action in a given state. It is an estimate of how good it is to choose a certain action in a specific situation.

To understand action-value, let's imagine a scenario where you are playing a game, and you need to make decisions to maximize your rewards. The action-value function would give you an estimate of how much reward you can expect to receive for each possible action you could take in a particular game state.

The book "Reinforcement Learning: An Introduction" by Sutton and Barto explains that the action-value function can be calculated using a formula called the Bellman equation. This equation takes into account the rewards you expect to receive for the current action, as well as the rewards you expect to receive for all future actions that you will take.[1]

The value of a state-action pair (s, a) is denoted by $Q(s, a)$ and is defined as the expected return (cumulative reward) that can be obtained by starting from the state s, taking the action a, and following the policy π thereafter. The expected return is defined as the sum of discounted future rewards, given by:[2]

$Q(s, a) = E [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s, A_t = a]$[2]

where R_{t+1} is the reward received at time step t+1, γ is the discount factor ($0 \leq \gamma \leq 1$), and the expectation is taken over all possible sequences of future rewards and states starting from time step t+1. The goal of action-value methods is to estimate the value function $Q(s, a)$ using experience (i.e., interactions with the environment), and use this estimate to improve the policy π.[2]

By using the action-value function, you can choose the action that is expected to give you the highest reward in a particular state. This is known as the "greedy" strategy. However, sometimes it is better to explore other options to gather more information about the game, and this is where the balance between exploration and exploitation comes into play.[1]

In summary, action-value is an estimate of the reward you can expect to receive for taking a particular action in a specific situation. It helps you make decisions to maximize your rewards in a reinforcement learning scenario.

References:

[1] https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[2] https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7