Multi-Armed Bandit

1. Introduction

1.1 Problem Statement

The multi-armed bandit problem is a fundamental problem in reinforcement learning. It models the dilemma between exploration (trying new actions to discover their potential rewards) and exploitation (choosing the known best action to maximize rewards). In this experiment, we analyse the performance of the -greedy algorithm with different values of in a stochastic multi-armed bandit setting.

The upcoming report will be divided into five sections: an explanation of the algorithm, an analysis of the relationship between exploration and exploitation, an examination of the action-value function, and validation.

Problem 1

2. Implementation (Code Explanation)

2.1 Multi-Armed Bandit Class

The `MultiArmedBandit` class simulates a k-armed bandit with rewards drawn from a normal distribution N(μ,1), where is sampled from N(0,1) . The estimated rewards are initialized with small noise to encourage exploration.

```
class MultiArmedBandit:
  def __init__(self, n_arms):
    self.n_arms = n_arms
    self.arms_true_reward = np.random.normal(0, 1, n_arms)
    self.count_arm = np.zeros(n_arms)
    self.estimated_reward = np.zeros(n_arms)
```

Equation for updating the estimated reward:

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$$

where:

- Q_{n+1} is the updated estimated reward.
- R_n is the received reward at step.
- n is the count of times action was selected.

2.2 Action Selection

The ε-greedy strategy chooses a random action with probability ε and the best estimated action with probability $1 - \varepsilon$.

```
  def select_action(self, greedy_rate):
    if np.random.random() < greedy_rate:
      return np.random.randint(self.n_arms)
      max_value = np.max(self.estimated_reward)
      max_indices = np.where(self.estimated_reward == max_value)[0]
      if len(max_indices) == 0:
        return np.random.randint(self.n_arms)
      return np.random.choice(max_indices)
```

The method decides which arm to pull (action to take) by balancing exploration (trying random arms to learn more) and exploitation (choosing the arm that currently seems best based on past results). And "tie-breaking" ensures the bandit doesn't always pick the same arm in a tie, giving a fair chance to all equally good options, ensuring the bandit learns the best arm over time while avoiding overfitting to early results or excessive randomness.

2.3 Reward Updating

Updating the estimated reward for the arm that was just chosen, based on the actual reward received.

```
  def update_reward(self, action, reward):

    self.count_arm[action] += 1
    n = self.count_arm[action]
    if n == 0:
      return
    self.estimated_reward[action] +=
    (reward - self.estimated_reward[action]) / n
```

By updating the estimated reward for each arm, more stable estimated reward over time, the bandit builds a picture of which arms are likely to give the highest rewards, allowing it to make smarter choices in the future.

2.4 Experiment Function

The experiment runs the bandit simulation and records per-step rewards and the frequency of optimal action selection.

```
def experiment(bandit, greedy_rate, n_steps):
  per_step_rewards = np.zeros(n_steps)
  optimal_actions = np.zeros(n_steps)
  optimal_arm = np.argmax(bandit.arms_true_reward)
  for i in range(n_steps):
    action = bandit.select_action(greedy_rate)
    reward = bandit.step(action)
    bandit.update_reward(action, reward)
    per_step_rewards[i] = reward
    if action == optimal_arm:
      optimal_actions[i] = 1
    else:
      optimal_actions[i] = 0
  return per_step_rewards, optimal_actions
```

At each step, an action is selected based on the provided strategy, a reward is obtained, and the bandit's internal estimates are updated. Performance metrics, specifically the reward obtained and an indicator of whether the optimal action was chosen, are recorded for subsequent analysis. Upon completion, the function returns these metrics, providing a comprehensive dataset for evaluating the strategy's performance. This function allows revealing the strategy's ability to accumulate rewards over time, assessing the accuracy and speed of learning the optimal choice, enabling a comparative analysis of different ε-greedy strategys.

3. Experiment and Results

3.1 Experiment

```
n_arms = 10
n_steps = 2000
greedy_rates = [0, 0.01, 0.1]
runs = 2000
results = {}

for greedy_rate in greedy_rates:
    all_per_step_rewards = np.zeros((runs, n_steps))
    all_opt_actions = np.zeros((runs, n_steps))
    for r in range(runs):
        bandit = MultiArmedBandit(n_arms)
        per_step_rewards, optimal_actions = experiment(bandit,
greedy_rate, n_steps)
        all_per_step_rewards[r, :] = per_step_rewards
        all_opt_actions[r, :] = optimal_actions
    avg_rewards = np.mean(all_per_step_rewards, axis=0)
    all_opt_actions = np.mean(all_opt_actions, axis=0)
    results[greedy_rate] = (avg_rewards, all_opt_actions)
```

Number of arms: 10

Total number of steps: 2000

Exploration rates: 0, 0.01, 0.1

Number of independent runs: 2000(Reduce variance in the results)

Within each run, a new instance of the MultiArmedBandit class is created. The experiment function is called to simulate the decision-making process over 2,000 steps, using the specified exploration rate. This function returns arrays of per-step rewards and optimal action indicators. After completing all 2,000 runs for a given exploration rate, the mean per-step rewards and mean optimal action indicators are calculated across runs.

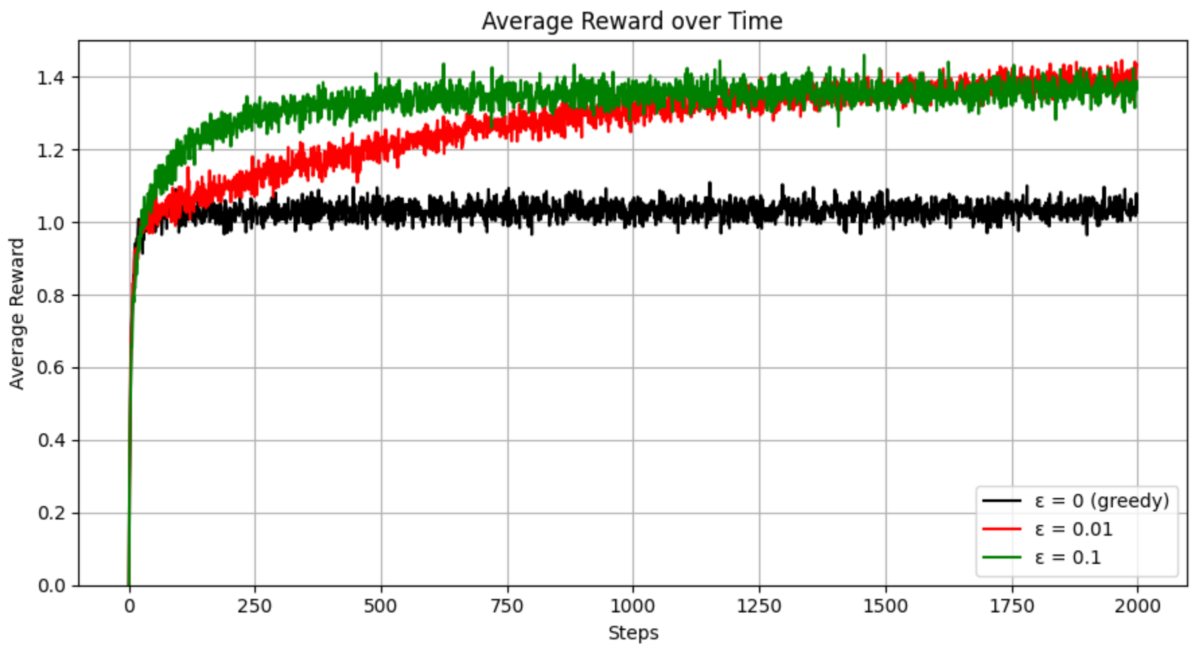
Next, analyse the experimental results.

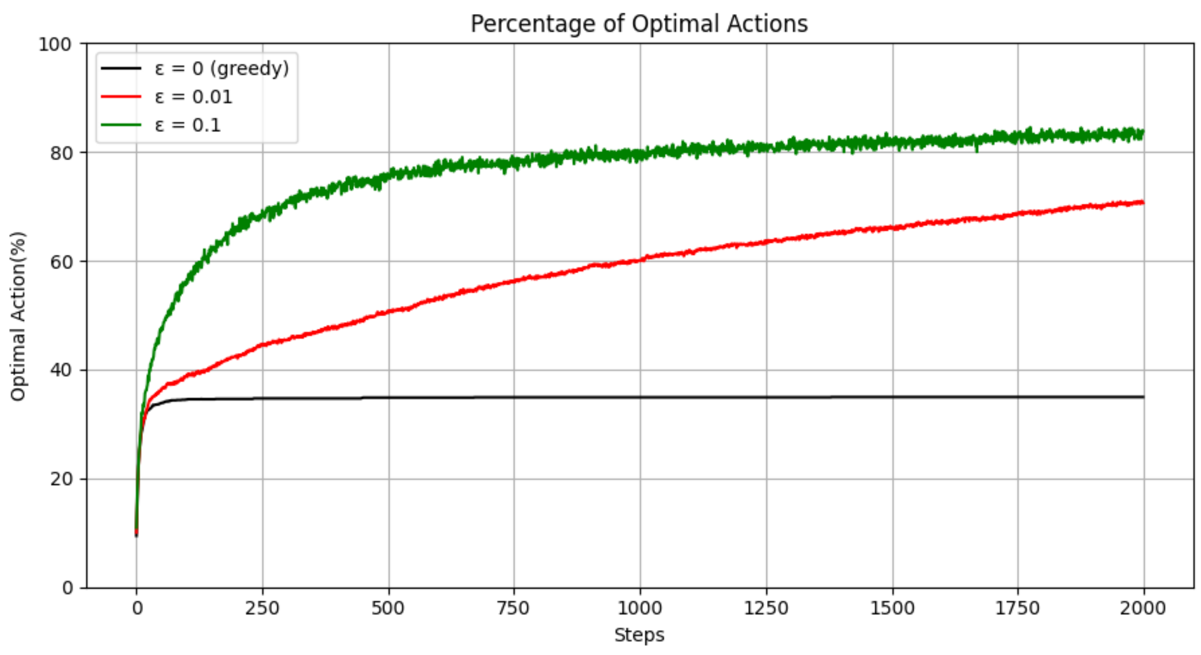
3.2 Results

Before discussing the results, re-analyse the purpose of this task：

This report analyses an experiment on Multi-Armed Bandit problems, focusing on the ε-greedy strategy's performance with different exploration rates (ε = 0, 0.01, 0.1). The experiment, conducted with 10 arms and 2000 steps per run over 2000 runs.

Metrics included final average reward, final optimal action percentage, and full experiment average reward.





The results showed:

```
Final Performance Metrics:
ε = 0: Final Avg Reward = 1.026, Final Optimal Action % = 34.95%
ε = 0.01: Final Avg Reward = 1.433, Final Optimal Action % = 70.60%
ε = 0.1: Final Avg Reward = 1.388, Final Optimal Action % = 83.80%

Full Experiment Average Rewards:
ε = 0: Full Avg Reward = 1.028
ε = 0.01: Full Avg Reward = 1.270
ε = 0.1: Full Avg Reward = 1.325
```

Preliminary Analysis:

- Fully Greedy (ε = 0): Without exploration, performance was poor, getting stuck on suboptimal arms, with low final and full average rewards.
- Low Exploration (ε = 0.01): Highest final average reward (1.433) due to minimal exploration noise at the end, but lower full average reward (1.270).
- Moderate Exploration (ε = 0.1): Highest full average reward (1.325) and optimal action percentage (83.80%), but slightly lower final reward (1.388) due to occasional exploration at step 2000.

An unexpected detail is that despite ε = 0.1 having a higher chance of choosing the optimal arm, its final average reward was lower than ε = 0.01, likely due to the 10% exploration rate occasionally selecting suboptimal arms at the last step.

Deeper Discussion:

1. **Fully Greedy Strategy (ε = 0):**

- Performance Metrics: Final Avg Reward = 1.026, Final Optimal Action % = 34.95%, Full Avg Reward = 1.028.
- Interpretation: Without exploration, the agent always exploits the arm with the highest estimated reward. Initially, all estimated rewards are zero, leading to random choices until one arm shows a higher reward by chance. The agent then sticks to that arm, potentially missing better arms. This results in getting stuck on suboptimal arms in many runs, explaining the low final average reward and optimal action percentage (only 34.95% of runs end with the optimal arm). The full average reward (1.028) is also low, reflecting poor overall performance due to lack of learning.

2. **Low Exploration (ε = 0.01):**

- Performance Metrics: Final Avg Reward = 1.433, Final Optimal Action % = 70.60%, Full Avg Reward = 1.270.
- Interpretation: With ε = 0.01, the agent explores rarely, allowing some learning about different arms but potentially insufficient to always identify the optimal arm. The final optimal action percentage (70.60%) indicates that in about 70.6% of runs, the agent ends with the optimal arm, but this is lower than ε = 0.1. However, the final average reward (1.433) is higher than ε = 0.1 (1.388), suggesting that in runs where it doesn't find the optimal arm, it may exploit a suboptimal arm with a relatively high true reward mean. The full average reward (1.270) is higher than ε = 0 but lower than ε = 0.1, indicating moderate long-term performance.

3. **Moderate Exploration (ε = 0.1):**

- Performance Metrics: Final Avg Reward = 1.388, Final Optimal Action % = 83.80%, Full Avg Reward = 1.325.
- Interpretation: With ε = 0.1, the agent explores more frequently, leading to better estimates of each arm's true reward. This is reflected in the highest final optimal action percentage (83.80%), meaning in 83.8% of runs, the agent ends with the optimal arm. The full average reward (1.325) is the highest, suggesting better long-term performance due to faster learning and identification of the optimal arm. However, the final average reward (1.388) is slightly lower than ε = 0.01 (1.433), likely because the 10% exploration rate occasionally selects suboptimal arms at step 2000, pulling down the final reward compared to ε = 0.01's minimal exploration noise.

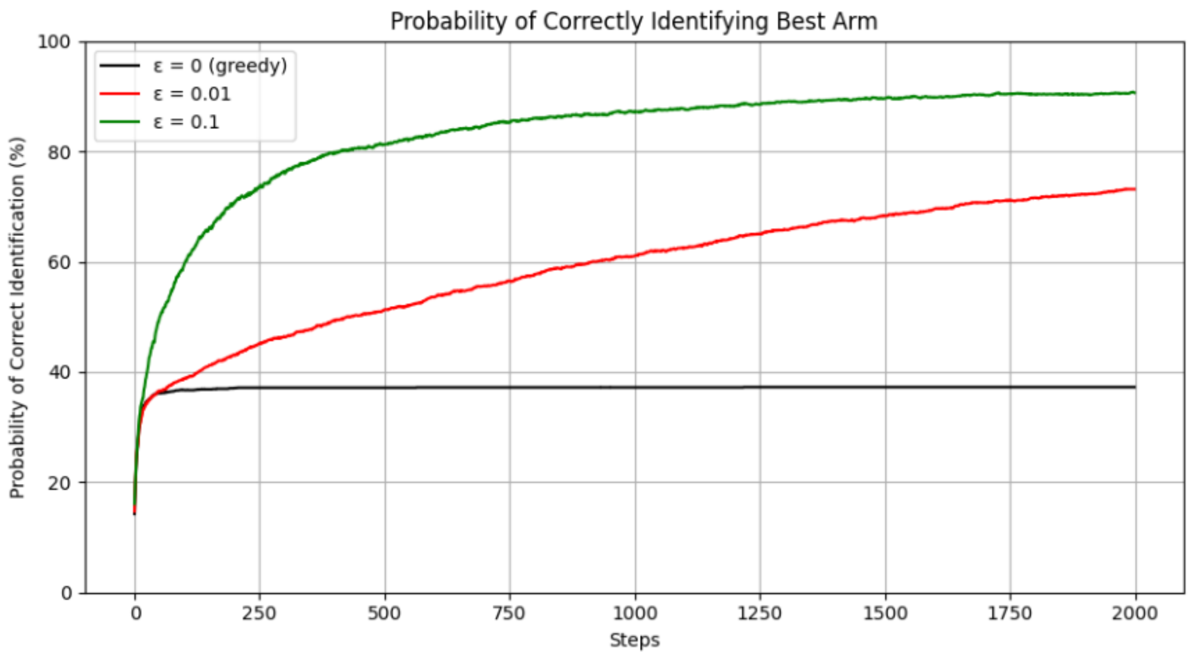
An interesting observation is that despite ε = 0.1 having a higher final optimal action percentage (83.80% vs. 70.60%), its final average reward is slightly lower than ε = 0.01. This will be discussed in the following exploration.

Problem 2

3. Exploration and Exploitation for Multi-armed Bandits

Before discussing the exploitation and exploration of multi-armed bandits, first think about the probability of identifying the best arm by choosing different exploration rates.

(I created a statistical chart for this, but didn't add it to my code.)



From this plot, the rapid increase in the probability of correctly identifying the best arm with ε = 0.1 underscores the power of exploration in accelerating learning. In contrast, the stagnation of this probability with ε = 0 highlights the peril of over-reliance on exploitation without sufficient data collection. This trade-off is not merely a function of ε but also depends on the problem's structure—such as the number of arms, reward variance, and the true reward distribution. In this experiment, with 10 arms and a normal reward distribution, the high variance and initial uncertainty necessitate exploration. The superior performance of ε = 0.1 in full experiment average reward (1.325) suggests that in environments with significant initial uncertainty, a moderate exploration rate is critical to uncover the optimal arm over time.

Analysing the trade-off between exploitation and exploration from regret. The result from ε = 0 in a near-constant regret over time, explaining the flat average reward curve, as the agent rarely updates its estimates beyond initial random choices. ε = 0.01is better but still some exploration regret, as the low exploration rate limits learning speed. The high final average reward indicates that, once a good arm is found, the agent exploits it effectively, minimizing late-stage exploration regret. For ε = 0.1, the rapid convergence to a high correct identification probability reflects low misidentification regret due to robust learning. However, the slight dip in final average reward compared to ε = 0.01 suggests ongoing exploration regret, as the 10% exploration rate occasionally selects suboptimal arms at the end.

This result also corresponds to the discussion on the exploration rate in the reference book, noting that the optimal ε value often decreases over time (e.g., ε ∝ 1/t, where t is the step number) to balance initial exploration with later exploitation. This adaptive approach could mitigate the trade-off observed here, potentially outperforming fixed ε strategies by reducing exploration regret in later stages while ensuring early learning. The theoretical analysis in the book shows that this method is particularly effective in a static environment because it collects information in the early stage and maximizes rewards in the later stage. Additionally, the literature mentions other strategies, such as UCB or Thompson Sampling, which may perform better in certain scenarios. However, ε-annealing (gradually reducing ε) is a simple and practical improvement.

Specifically, using a higher exploration rate (e.g., ε = 0.1) in the early stages accelerates learning, enabling the agent to quickly identify the optimal arm. However, in the later stages, a lower exploration rate (e.g., ε = 0.01) is more effective, as it minimizes exploration noise and maximizes exploitation of the best-known arm. This finding insight that an optimal ε often decreases over time (e.g., ε ∝ 1/t) to balance initial exploration with later exploitation, reducing regret while ensuring robust learning.

In summary, the experiment suggests adopting a dynamic exploration strategy: start with a higher exploration rate to expedite learning and identify the optimal arm, then transition to a lower exploration rate to focus on exploitation and maximize rewards in the later stages. This approach optimizes both long-term learning and final performance in stationary environments of bandit.

So, in stationary environments the optimal strategy for balancing exploration and exploitation in Multi-Armed Bandit problems is to adopt an exploration-decreasing approach.

Problem 3

4. Action-value Methods

Action-value methods are a technique in reinforcement learning used to estimate the expected return of each action. In the multi-armed bandit problem, each "arm" represents an action, and we need to estimate the expected reward of pulling each arm. My code uses the sample average method, which updates the estimated value by computing the average reward for each arm.

```
def update_reward(self, action, reward):

    self.count_arm[action] += 1
    n = self.count_arm[action]
    if n == 0:
      return
    self.estimated_reward[action] +=
    (reward - self.estimated_reward[action]) / n
```

In the code, after pulling an arm, the reward is generated based on a normal distribution, and the estimated value is updated using the following formula:

```
self.estimated_reward[action]+=(reward-self.estimated_reward[action])/n
```

Where n is the number of times the arm has been selected. This method ensures that the estimated value gradually converges to the true expected reward as more samples are collected. The true reward for each arm is sampled from N (0,1) and remains fixed. Each time an arm is pulled, the reward is generated from N({true_reward}, 1).The estimated values are initialized to zero and updated incrementally. This method is efficient in a stationary environment because it ensures an unbiased estimate, and as the sample size increases, the variance decreases.

4.1 Analysis of Other Different Action-Value Methods

4.1.1 Fixed Learning Rate Method

Instead of using the sample average, a fixed learning rate α can be used, updating the estimate as:

```
self.estimated_reward[action]+=α*(reward-self.estimated_reward[action])
```

where α is a constant (e.g., 0.1). This method weights recent rewards more heavily, introducing volatility in estimates. In a stationary environment, this can lead to higher variance compared to sample average, potentially causing the agent to switch arms more frequently, mimicking additional exploration. For ε = 0, where the agent risks getting stuck on suboptimal arms, a larger α (e.g., 0.5) might allow estimates to drop quickly with poor rewards, increasing the chance of switching to other arms with stale estimates, thus improving performance. However, in the long run, fixed learning rates may not converge as accurately as sample averages, potentially reducing final average rewards in high-ε scenarios.

4.1.2 Bayesian Estimation

Bayesian methods assume a prior distribution for each arm's true reward and update it with observed rewards to form a posterior distribution.

This is akin to a smoothed sample average, pulling estimates towards zero initially. For ε = 0, this conservatism might prevent early locking onto arms with slightly positive rewards, as estimates are less likely to spike high with few samples, potentially improving the chance of exploring other arms later. In high-ε scenarios, the impact may be minimal, as explicit exploration dominates.

4.1.3 Other Statistical Methods

Using the median instead of the mean for estimation could be considered, but given rewards follow a normal distribution, the mean is optimal, and median would yield similar results. Robust estimators for outliers are unnecessary here, as the reward distribution is well-behaved.

4.2 Discussion for Action-Value Methods

Although the sample estimation method seems to be suitable for 10-armed Bandit, we can still consider multiple options when the exploration rate is different.

In the current experiment, when exploration rates are low, like ε = 0, the agent mostly exploits, relying on current estimates, which can lead to getting stuck on suboptimal arms.

In contrast, a fixed learning rate (e.g., updating with a constant step size) introduces variability. This means even after many pulls, the estimate can fluctuate with new rewards, potentially dropping below other arms' estimates (which might be zero if unexplored). This variability can help the agent switch to better arms, mimicking some exploration, which is crucial when explicit exploration (ε) is low. For example, if a suboptimal arm gives a low reward, its estimate drops faster, possibly leading the agent to try another arm.

Interestingly, while fixed learning rates help in low exploration, they don't converge to the true reward like sample averages, introducing a trade-off between short-term adaptability and long-term accuracy. This means in the experiment, with ε = 0.01 or 0.1, where exploration is already higher, the sample average

$$Q(a)_{n+1} = Q(a)_n + \frac{1}{n}(R_n - Q(a)_n)$$

might still be better for overall performance.

But for low exploitation, If the agent gets stuck on a suboptimal arm, its estimate Q(a) converges to the true reward R, which may be less than the best arm's reward. Other arms, if unexplored, have estimates at their initial value (zero in the code), and the agent remains stuck.

With a fixed learning rate

$$Q(a)_{n+1} = Q(a)_n + \alpha(R_n - Q(a)_n)$$

even after many pulls, Q(a) can fluctuate due to random rewards. For instance, if a suboptimal arm gives a reward R<Q(a), the estimate decreases, potentially below zero, making another arm with estimate zero look better. This fluctuation can lead to switching arms, providing a chance to discover better arms.

This variability acts as implicit exploration, crucial when explicit exploration (ε) is low.

In stationary environments, sample average is optimal for convergence. However, in low ε scenarios, the agent's adaptability is limited. Fixed learning rates, often used in non-stationary settings, introduce higher variance, which can be beneficial for escaping local optima in low exploration contexts, as discussed in Multi-armed Bandit Problem in Reinforcement Learning. While fixed learning rates help in low ε, they sacrifice long-term accuracy. The steady-state variance means estimates never settle, potentially leading to worse performance in high ε scenarios where exploration already ensures frequent updates. In low exploration rates, changing to a fixed learning rate for action-value estimation can be more beneficial than the sample average method, as it introduces variability, helping the agent switch from suboptimal arms and discover better ones. This implicit exploration is crucial when explicit exploration is limited, though it trades off long-term accuracy for adaptability.

The conclusion is that in stable environments with high exploration, the sample average method is optimal, capitalizing on the extensive data gathered to achieve accurate and stable estimates, leading to the best long-term performance.

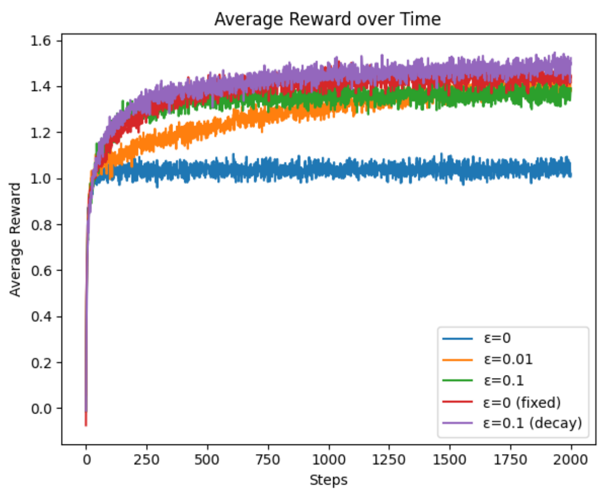
Use a fixed learning rate for low exploration to introduce variability and escape suboptimal arms, improving early performance despite reduced convergence, overcoming exploration deficiencies.

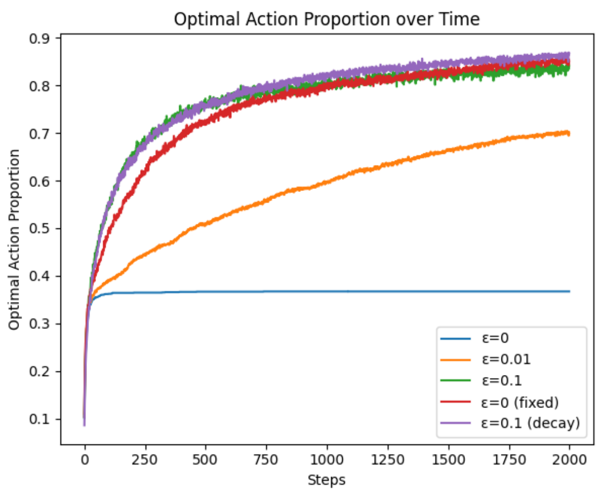
5. Verification

After completing the analysis of various exploration rates, I used technical means to verify the conclusions drawn.

Using a fixed exploration rate of 0.05 based on ε=0.

Based on ε=0.1, a gradually decreasing exploration rate is used, with an initial value of 0.1 and a decay rate of 0.001.





Conclusion:

ε = 0 (pure greed) performs the worst, indicating that not exploring at all leads to the algorithm getting stuck in a local optimum, resulting in a lower ratio of rewards and optimal actions.

A higher fixed exploration rate (e.g., ε = 0.1) is more effective than a lower exploration rate (e.g., ε = 0.01), but it may cause reward fluctuations due to excessive exploration.

Advantages of decaying exploration rate: ε = 0.1 (decay) performs the best in terms of reward and optimal action ratio, suggesting that a high initial exploration rate helps discover the optimal arm, while gradually decreasing the exploration rate allows the algorithm to make more efficient use of the acquired information.

Fixed exploration rate: ε = 0 (fixed) offers a compromise, with performance between pure greed and higher exploration rates, making it suitable for scenarios that require a balance between exploration and exploitation. This also confirms the previous analysis and research from the reference book.

(Since this is not part of the task, I did not include these codes in my notebook.)