Introduction to Artificial Intelligence Homework 4

112550139 簡士原

1. Implementation Details

Environment:

I implemented a k-armed Gaussian bandit environment. To achieve this, I implemented 3 functions required by the PDF file:

- reset(): Initialize the bandit with μ sampled from $\mu_i \sim N(0,1)$ and clear the logs of actions and rewards.
- step(): Execute one pull of the selected arm and return the observed reward.
  - For the stationary case, each arm's reward is drawn from a Gaussian distribution $N(\mu_i, 1)$.
  - For the non-stationary cases, which means stationary=False, all μ will perform a Gaussian random walk at every time step, i.e., $\mu_i \leftarrow \mu_i + \epsilon, \epsilon \sim N(0, 0.01^2)$.
- export_history(): Return all past actions and rewards.

Agent:

I follow an ε-greedy action-value strategy, utilizing 3 methods:

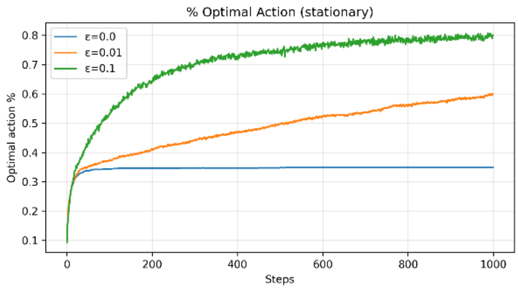
- reset(): Setting estimate vector Q and counter N to all 0s.
- select_action(): Choose an arm according to an ε-greedy policy.
  - Exploration: With probability **ε,** select a random arm.
  - Exploitation: Otherwise, choose the best action where an arm whose current estimate Q_i is maximal. If there are multiple best actions, randomly choose one.
- update_q(): Update the value estimate of the chosen arm, a, just pulled.
  - Learning rate η: If a fixed learning-rate α is provided, use η = α. Otherwise, use the sample-average rate: $\eta = \frac{1}{N_a}$.
  - Update method:

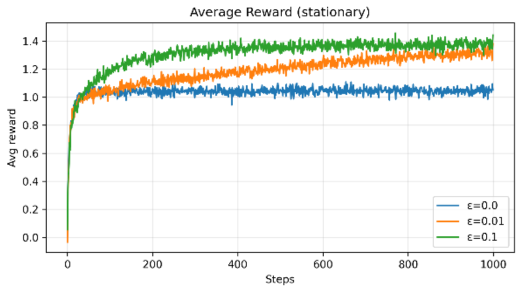
    $Q(a) \leftarrow Q(a) + \eta \cdot (R - Q(a))$.

    Increment counter N_a by 1.

2. Experiment Results

- Part 3: Setting stationary=True, sample-average, ε={0, 0.01, 0.1}



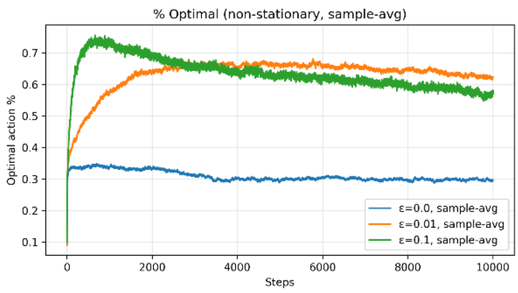


A. Pure exploitation (ε=0.0) quickly plateaus on a suboptimal arm, so both its average reward and optimal-action rate stagnate at the lowest level.

B. Moderate exploration (ε=0.01) discovers the optimal arm more reliably than ε = 0 while barely sacrificing early reward, leading to higher final reward and optimal-action percentage.

C. High exploration (ε=0.1) finds the optimal arm fastest, giving the highest optimal-action rate, and ultimately surpasses ε = 0.01 in average reward once the cost of exploration is offset.

- Part 5: Setting stationary=False, sample-average, ε={0, 0.01, 0.1}

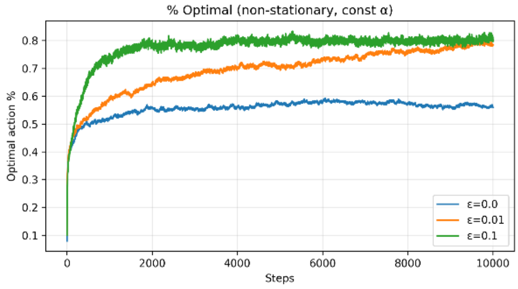


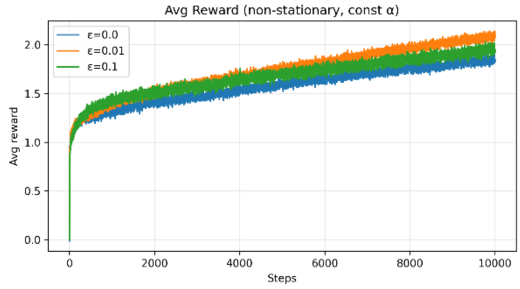
A. Pure exploitation (ε = 0) cannot rediscover arms whose means have drifted, so both its average reward and optimal-action rate stay low and even decline.

B. High exploration (ε = 0.1) reacts quickly at first, achieving the highest early optimal-action rate, but its constant probing adds noise. Combined with the slow-adapting sample-average estimate, this causes a gradual drop in optimal-action rate and leaves its final reward below that of ε = 0.01.

C. Moderate rate (ε = 0.01) strikes the best balance: it refreshes knowledge often enough to track the drifting optimum, yet avoids the excessive variance of ε = 0.1, ending with the highest average reward and the most sustained optimal-action percentage.

- Part 7: Setting stationary=False, α=0.1, ε={0, 0.01, 0.1}





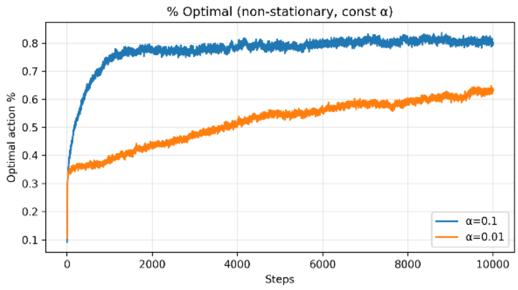
A. Pure exploitation (ε = 0): With α fixed at 0.1, every new reward quickly influences the value estimate, so even ε = 0 can climb well above its Part-5 performance; but without exploration, it levels off lowest in both metrics.

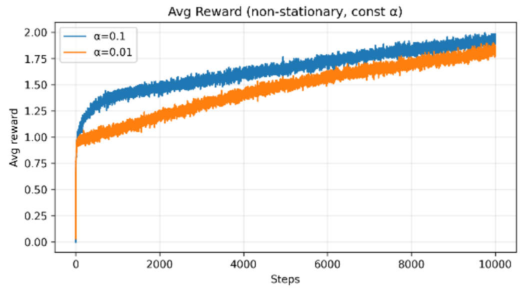
B. High exploration (ε = 0.1) keeps finding the current best arm fastest, giving the top optimal-action percentage (≈ 0.80). However, its frequent random pulls add variance, so its average reward lags slightly behind ε = 0.01.

C. Moderate exploration (ε=0.01) strikes the best balance: enough exploration to track drifting optima, yet few enough random pulls that its average reward ends highest while its optimal-action rate stays close to that of ε = 0.1.

3. Discussion:

I ran one more little test just accidentally: kept ε = 0 and only changed the step-size α. Turns out α = 0.1 outperforms α = 0.01 significantly.





I think it's all about how fast the Q-values can 'forget' old rewards: with α = 0.1, the agent's memory only stretches back about 10 pulls, so it notices the drift almost right away and jumps to the new best arm. In contrast, α = 0.01 clings to a 100-step history, basically sleeping through the change and sticking to a now-suboptimal arm for ages.