Multi-armed bandits

↗ Course	📙 Advanced Machine Learning

Introduction to Multi-Armed Bandits

Multi-armed bandits (MAB) are a fundamental concept in reinforcement learning and decision theory. The name comes from slot machines (one-armed bandits) - imagine having multiple slot machines and trying to maximize your winnings by choosing which machine to play.

Core Concept

The multi-armed bandit problem represents the trade-off between exploration and exploitation:

- Exploration: Trying different options to gather information
- Exploitation: Using current knowledge to maximize rewards

Key Components

A multi-armed bandit problem consists of:

- Multiple "arms" (actions or choices)
- Unknown probability distributions of rewards
- Sequential decision-making process
- Feedback only for the chosen action

Common Strategies

Several strategies exist to solve MAB problems:

- ε-greedy: Choose the best-known action with probability 1-ε, and explore randomly with probability ε

- Upper Confidence Bound (UCB): Balance exploration and exploitation using confidence intervals
- Thompson Sampling: Use Bayesian inference to select actions based on probability matching

Applications

Multi-armed bandits are widely used in:

- A/B Testing and Website Optimization
- Online Advertising
- Clinical Trials
- Resource Allocation
- Recommendation Systems

Mathematical Formulation

The goal is to maximize the cumulative reward over time T:

$$R(T) = \sum_{t=1}^{T} r_t$$

Where r_t is the reward at time t. The challenge is to minimize the regret, which is the difference between the optimal strategy and the chosen strategy.

A k -armed Bandit Problem

After each choice you receive a numerical reward chosen
from a stationary probability distribution that depends on the action you selected

your objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or time steps.

In our k -armed bandit problem, each of the k actions has an expected or mean reward
given that that action is selected; let us call this the value of that action.

We denote the action selected on time step t as At, and the corresponding reward as Rt. The value then of an arbitrary action a, denoted q⇤(a), is the expected reward given that a is selected:

$$q*(a) = E[Rt|At = a]$$

We denote the estimated value of action a at time step t as Qt(a). We would like Qt(a) to be close to $q*(a)$.

▼ What is greedy actions?

If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call these the greedy actions.

▼ What is exploiting and exploring?

Selecting the greedy action means exploiting your current knowledge for immediate reward, while selecting non-greedy actions means exploring to improve your knowledge for potentially better rewards in the future.

Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run.

Action-Value Methods

Action-value methods are ways to estimate the values of actions and use these estimates to make action selection decisions.

▼ What is Sample-Average Method?

The most common action-value method is to estimate the value of an action by averaging the rewards actually received when that action was selected:

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to t}}{\text{number of times a taken prior to t}}$$

If the denominator is zero, then we define Qt(a) as some default value, such as 0.

▼ How to implement ε-greedy action selection?

When using ε-greedy action selection, most of the time we select an action that has maximal estimated value (breaking ties randomly), but with small probability ε, we instead select an action at random.

- With probability 1-ε: Select action with highest estimated value
- With probability ε: Select a random action

This ensures continued exploration while still favoring actions that appear to be best.

The advantage of action-value methods is that they are simple to implement and understand, while still providing effective performance in many practical applications.

⁉️ Exercise 2.1: In ε-greedy action selection, with two actions and ε = 0.5, what is the probability that the greedy action is selected?

💡 Let's solve this step by step:

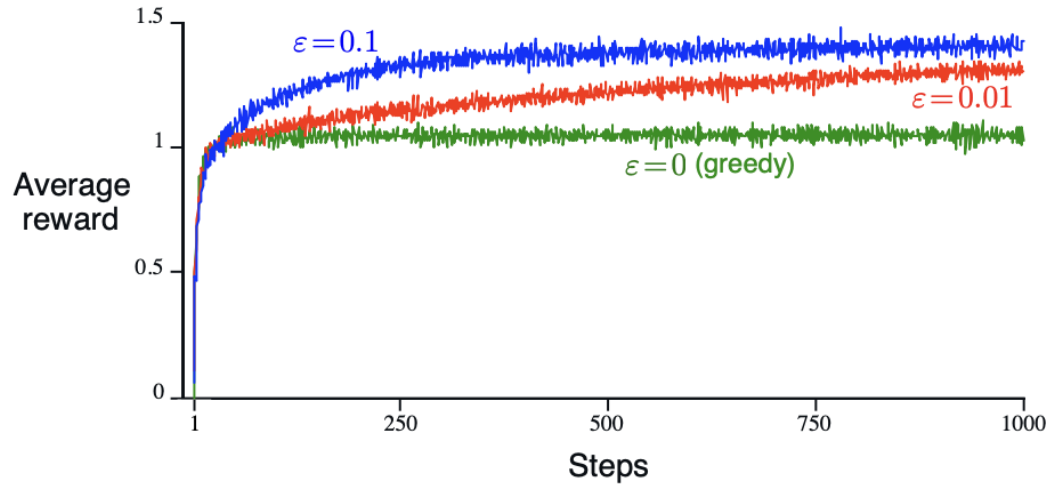
- The probability of selecting the greedy action comes from two scenarios:

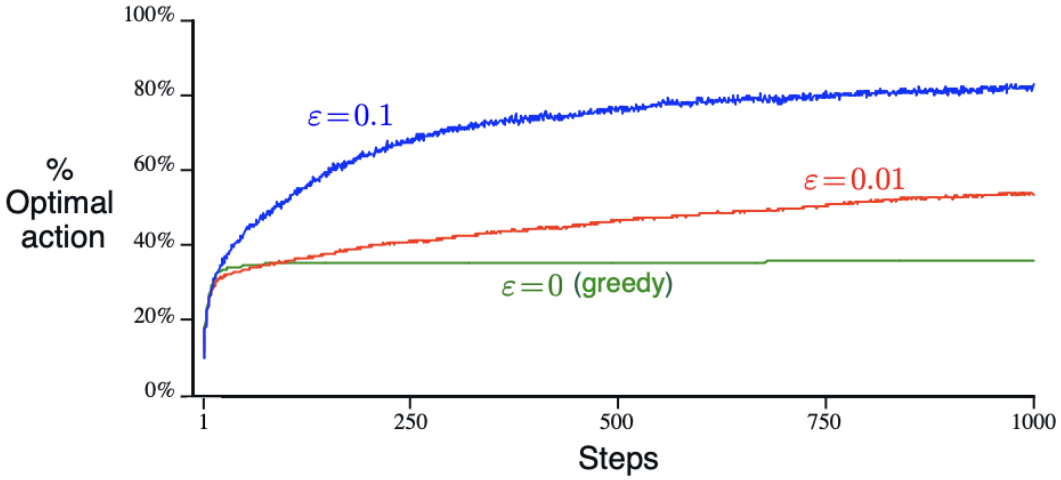
1. When we exploit (probability 1-ε = 0.5), we definitely choose the greedy action
2. When we explore (probability ε = 0.5), we randomly choose between the two actions, so we have a 0.5 chance of selecting the greedy action

Therefore, the total probability is:

$$P(\text{greedy}) = (1 - \varepsilon) + \varepsilon \cdot \frac{1}{2} = 0.5 + 0.5 \cdot 0.5 = 0.75$$

The answer is 0.75 or 75%.





The Advantage of ε-Greedy Over Greedy Methods

- The relative performance of ε-greedy vs. greedy depends on the task conditions:
  - **Noisy Rewards**:
    - If the reward variance (how much the observed rewards fluctuate) is large, more exploration (ε>0) is necessary to accurately identify the optimal action.
    - For instance, if reward variance is 10 (noisier), ε-greedy will outperform greedy because it allows the agent to explore more and avoid being misled by random fluctuations.
  - **Deterministic Rewards**:

- If the rewards have zero variance (perfectly predictable), greedy methods perform well because the agent can identify the optimal action with a single trial and then exploit it.
- However, even in deterministic tasks, $\varepsilon > 0$ may be advantageous in certain conditions, such as nonstationarity.

  $\varepsilon > 0$

Nonstationary Tasks

- **Nonstationary Environment**:
  - In many real-world scenarios, the optimal action changes over time (e.g., due to changing conditions in the environment). This is called a nonstationary task.
  - Greedy methods, which stop exploring once they find an optimal action, may fail in nonstationary tasks because they don't adapt to changes in action values.
  - $\varepsilon\varepsilon$-greedy methods can perform better because exploration allows the agent to periodically revisit suboptimal actions, ensuring they haven't become optimal over time.

Stationary vs. Nonstationary in Reinforcement Learning

- Even in stationary tasks, reinforcement learning often involves dynamic scenarios where the agent's policy evolves over time.
- Exploration is crucial in these cases to maintain a balance between exploiting known information and adapting to changes in the task structure.

❓ Exercise 2.2: Consider a 4-armed bandit problem with actions labeled 1, 2, 3, and 4. We apply an ε-greedy algorithm using sample-average action-value estimates, with initial estimates Q1(a) = 0 for all actions. Given this sequence of actions and rewards:
A1 = 1, R1 = -1
A2 = 2, R2 = 1
A3 = 2, R3 = -2
A4 = 2, R4 = 2
A5 = 3, R5 = 0

The ε case (random action selection) may have occurred on some steps. On which time steps did this definitely happen? On which steps could it possibly have happened?

▼ Solution

he agent selects actions either greedily (choosing the action with the highest estimated value) or randomly (exploration with probability ε).ε

- Initially, all actions have the same value (Q(a)=0), so the agent could pick any action randomly.Q(a)=0

Analysis:

1. **Step 1 (A_1=1)**:
    - Since all Q(a)=0, this could be either a random choice or a tie-breaking greedy choice.

      Q(a)=0
    - **Possible exploration**.
2. **Step 2 (A_2=2)**:
    - After Step 1: Q(1)=-1, Q(2)=Q(3)=Q(4)=0.

      Q(1)=-1

      Q(2)=Q(3)=Q(4)=0

- A_2=2 could be random or greedy (as Q(2) is among the highest values).

  Q(2)

- **Possible exploration**.

3. **Step 3 (A_3=2)**:
   - After Step 2: Q(1)=-1, Q(2)=1, Q(3)=Q(4)=0.

     Q(1)=-1

     Q(2)=1

     Q(3)=Q(4)=0

   - A_3=2 is greedy (as Q(2) has the highest value).

     Q(2)

   - **No exploration**.
4. **Step 4 (A_4=2)**:
   - After Step 3: Q(2)=-0.5, Q(1)=-1, Q(3)=Q(4)=0.

     Q(2)=-0.5

     Q(1)=-1

     Q(3)=Q(4)=0

   - A_4=2 must have been selected through exploration, as it's not the greedy choice.
   - **Definite exploration**.
5. **Step 5 (A_5=3)**:
   - After Step 4: Q(2)=0.33, Q(1)=-1, Q(3)=Q(4)=0.

     Q(2)=0.33

     Q(1)=-1

     Q(3)=Q(4)=0

   - A_5=3 could be either greedy (tie-breaking) or exploratory.
   - **Possible exploration**.

Answer:

- **Definite exploration**: Step 4.
- **Possible exploration**: Steps 1, 2, and 5.

❓ Exercise 2.3: Referring to Figure 2.2, which method will achieve the best long-term performance in terms of cumulative reward and probability of selecting the optimal action? Quantify the expected performance difference.

▼ Solution

- **Performance Comparison**:
  - Greedy:
    - Performs poorly in the long run because it doesn't explore.
    - If it initially selects a suboptimal action, it will never learn about better actions.
  - ε-Greedy:
    - Performs better as it balances exploration and exploitation.
    - Ensures that the optimal action is eventually identified with high probability.
- **Quantitative Answer**:
  - For small ε (e.g., ε=0.1):
    - The probability of selecting the best action in the long run approaches 1 as the agent learns the optimal action value.
    - The cumulative reward is higher than greedy because the agent avoids getting stuck with suboptimal actions.
  - The improvement depends on the specifics of the reward distributions and the number of steps. Exact improvement can be simulated for a specific testbed setup.

Incremental Implementation

- Instead of storing all rewards, we can update Qn incrementally when a new reward Rn is received.
- The update formula:

  $$Qn + 1 = Qn + 1/n[Rn - Qn]$$

  - Qn: The current estimate of the action value
  - Rn: The most recent reward
  - 1/n: The step size, which decreases as n increases
  - [Rn − Qn]: The difference between the new reward and the current estimate (the error)

Advantages of the Incremental Formula

- **Memory efficiency**: Only needs to store two values—Qn (the current estimate) and n (the action selection count).
- **Computational efficiency**: Updates require the same amount of computation at each step, regardless of how many trials have occurred.

$$NewEstimate \leftarrow OldEstimate + StepSize \times [Target - OldEstimate]$$

Intuition Behind the Update Formula

1. **Target - OldEstimate**:
   - This is the **error** in our current estimate. It measures the difference between our estimate (Qn) is from the new observed reward (Rn).
2. **StepSize**:
   - Controls how much we adjust our estimate. A larger step size leads to faster adjustments based on new information.
3. **NewEstimate**:
   - Adjusts toward the target reward, with updates becoming smaller as n increases (since 1/n decreases). This creates a learning process where early observations have stronger influence, while later updates become more gradual as the estimate stabilizes.

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$