Multi-Armed Bandit

Minor in AI - IIT ROPAR

15th April, 2025

Brewing the Best: A Quest for the Perfect Cup

Alex had just moved to a new city and was on a mission: to find the best coffee in town. There were five coffee shops, each with a different chance of offering the perfect cup. The challenge? Balancing exploration (trying new places) and exploitation (sticking with the best-known option).

Alex began at "Brewed Bliss," but the coffee was just average. They tried "Café Dream," and while it was better, it still wasn't great. The clock was ticking, and Alex had to decide: explore more or settle on one of the shops they'd already visited?

This dilemma was like the multi-arm bandit problem, a puzzle about making decisions with limited information. Should Alex risk trying more shops, hoping to find the best one? Or should they stay and enjoy a "good enough" cup at the best-known option?

In the end, Alex embraced exploration. They ventured to "Espresso Escape," and at last, discovered the rich, bold espresso they'd been searching for. It was the perfect balance of adventure and reward, teaching Alex that sometimes, the right choice isn't about sticking to what you know — it's about finding what's truly the best.

]

Now that we have introduced the multi-armed bandit (MAB) problem through example, let us explore how to tackle this problem.

The multi-armed bandit problem is to design a sequence of actions that will maximize the expected total reward over some time period.

The challenge lies in the uncertainty of reward distributions. Since the outcome of each arm (action) is uncertain, and we do not know in advance which arm (action) yields the best outcome (reward), our goal is to learn over time and gradually identify which arms (actions) tend to yield better outcomes.

From Uncertainty to Estimates

In the MAB problem, each arm (action) corresponds to an unknown probability distribution of rewards. Each time we choose an arm (action), we receive one sample from this distribution. Ideally, if we had complete knowledge of these reward distributions, we could simply pick the arm (action) with the highest expected reward at each step.

However, since we don't, we need a way to quantify our experience with each arm (action) using just a single, informative number — a number that summarizes what we know about that arm's reward behavior so far.

That number is the **mean**.

Why Use the Mean?

The mean is the simplest and most intuitive estimate for the expected reward from an action. It captures the central tendency of observed outcomes and becomes more accurate with more samples.

$$q(a) = \mathbb{E}[R_t \mid A_t = a]$$
$$Q_t(a) = \text{Estimate of } q(a) \text{ at time } t$$

The most natural and intuitive solution is to use the average of the rewards, that is, the mean (or expected reward) observed so far. It captures the central tendency of observed outcomes and becomes more accurate with more samples.

Let us understand why:

- Each arm produces a sequence of rewards.
- These rewards are samples from an unknown distribution.
- The true value of an action a is defined as the expected reward when that action is selected:

$$q(a) = \mathbb{E}[R_t \mid A_t = a]$$

This is a theoretical quantity — we don't know it in practice. So instead, we approximate it from the rewards we actually observe.

Estimating Action Values

As described in Sutton & Barto:

We denote the true value of action a as $q(a)$, and the estimated value on the t-th time step as $Q_t(a)$. One natural way to estimate this is by averaging the rewards actually received when the action was selected.

If by the t-th time step, action a has been chosen $N_t(a)$ times prior to t, yielding rewards $R_1, R_2, ..., R_{N_t(a)}$, then its value is estimated to be:

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)} = \frac{1}{N_t(a)} \sum_{i=1}^{N_t(a)} R_i$$

This equation defines the sample mean of all the rewards obtained by taking action a up to time t. It is the foundation of most algorithms for estimating action values in the multi-armed bandit setting.

Practical Motivation

Let us look at a simple numerical example:

Suppose we have pulled a particular arm a three times and received the following rewards:

$$R_1 = 2, \quad R_2 = 5, \quad R_3 = 3$$

Then the estimated value is:

$$Q(a) = \frac{2+5+3}{3} = \frac{10}{3} \approx 3.33$$

This value becomes our current belief about the expected reward from choosing this action. In other words, we treat it as the best guess of how good that arm is, based on what we've seen so far.

Every time we play that arm again and observe a new reward, we update this estimate. This helps us rank our options, compare actions, and gradually move toward more rewarding decisions as learning progresses.

Alternative Form: Using the Indicator Function

We can express this more formally using an indicator function:

$$Q_t(a_i) = \frac{\sum_{j=1}^{t} \mathbf{1}\{A_j = a_i\} R_j}{\sum_{j=1}^{t} \mathbf{1}\{A_j = a_i\}}$$

What does the indicator function mean?

$$\mathbf{1}\{A_j = a_i\} = \begin{cases} 1, & \text{if action } a_i \text{ was selected at time } j \\ 0, & \text{otherwise} \end{cases}$$

So:

- The numerator sums up rewards received only when a_i was played.
- The denominator counts how many times a_i was selected.

Formal Notation and Setup

Let us define a few notations clearly:

- N: Number of actions (arms)
- a_i: The ith action
- A_t: Action selected at time t
- R_t: Reward received at time t
- $q(a)$: True value of action a
- $Q_t(a)$: Estimated value of action a at time t
- $N_t(a)$: Number of times action a has been selected up to time t

Estimating the Value from Observed Rewards

Now that we understand how the value of an action can be estimated using observed rewards, we summarize this process with a practical lens.

Given our interaction history up to time t, we can compute the empirical value Qt(ai) of each action ai using the following formula:

$$Q_t(a_i) = \frac{\sum_{j=1}^{t} \mathbf{1}\{A_j = a_i\} R_j}{\sum_{j=1}^{t} \mathbf{1}\{A_j = a_i\}}$$

- The numerator sums rewards when a_i was selected.
- The denominator counts how many times a_i was selected.
- If the denominator is 0, initialize $Q_t(a_i)$ to 0 or some optimistic value.

This empirical average serves as the foundation for decision-making algorithms in the multi-armed bandit setting.

In the next subsection, we demonstrate how this plays out through a concrete example.

Illustrative Example

Suppose at time steps 1 through 4:

$$\begin{aligned} A_1 &= a_1, & R_1 &= 1 \\ A_2 &= a_1, & R_2 &= 2 \\ A_3 &= a_2, & R_3 &= 5 \\ A_4 &= a_1, & R_4 &= 4 \end{aligned}$$

At $t = 5$:

$$\begin{aligned} N_5(a_1) &= 3, & Q_5(a_1) &= \frac{1+2+4}{3} = \frac{7}{3} \approx 2.33 \\ N_5(a_2) &= 1, & Q_5(a_2) &= \frac{5}{1} = 5 \end{aligned}$$

Deriving the Recursive Update Rule

In the long run, storing all past rewards to compute averages becomes impractical. Instead, we use an incremental update rule that avoids full memory of the reward history and lets us update the value estimate using only the most recent reward and the current estimate.

Storing all past rewards is impractical. So, we use an incremental update rule:

Let Q_n be the estimate after n observations and R_n the reward at step n. The sample mean is:

$$Q_n = \frac{1}{n} \sum_{i=1}^{n} R_i$$

Expressing Q_n in terms of Q_{n-1}:

$$\begin{aligned} Q_n &= \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) \\ &= \frac{1}{n}(R_n + (n-1)Q_{n-1}) \\ &= Q_{n-1} + \frac{1}{n}(R_n - Q_{n-1}) \end{aligned}$$

This update rule allows for incremental computation of the mean using only the previous estimate and the most recent reward.

The Multi-Armed Bandit (MAB) problem presents a fundamental dilemma: how to make decisions under uncertainty while balancing exploration (trying new options) and exploitation (choosing the best-known option).

A natural starting point is the Greedy Algorithm. As the name suggests, it always selects the arm with the highest observed average reward. Initially, each arm is played once to get a rough idea of its potential. After that, the greedy algorithm continually selects the best-performing arm so far.

Drawbacks of the Greedy Approach

- It may converge too early to a suboptimal arm, ignoring better options.
- It completely avoids exploring arms with poor initial results.
- It is highly sensitive to initial randomness and noise in rewards.

To overcome these issues, we introduce randomness into decision-making — leading to the Epsilon-Greedy algorithm.

Epsilon-Greedy Algorithm

The Epsilon-Greedy algorithm builds upon the basic greedy strategy by introducing a small degree of randomness to encourage exploration alongside exploitation.

At the heart of this method is a parameter called epsilon (ε), which represents the probability of exploring — that is, selecting an arm at random instead of choosing the one that currently appears to be the best.

- With probability ε, the algorithm selects an arm at random (exploration).
- With probability $1 - \varepsilon$, it selects the arm with the highest estimated average reward so far (exploitation).

Here's how it works at each time step:

1. A random number between 0 and 1 is generated.
2. If this number is less than ε, the algorithm explores by choosing an arm randomly.
3. Otherwise, it exploits by selecting the arm with the highest current average reward estimate.

The value of the hyperparameter $\varepsilon \in [0, 1]$ controls how often exploration occurs. A smaller ε (such as 0.1) means the algorithm mainly exploits, while still occasionally exploring new options.

This approach is beneficial because it:

- Continues to prioritize high-performing arms (exploitation),
- While still giving lower-performing or less-tested arms a chance to be evaluated (exploration).

Tuning ε is essential:

- A small value (e.g., $\varepsilon = 0.1$) allows mostly greedy behavior with occasional exploration.
- A larger value (e.g., $\varepsilon = 0.5$) increases the rate of exploration, which can be especially useful during the early stages of learning.

Over time, this method helps build accurate estimates of each arm's reward probability. By effectively balancing exploration and exploitation, the epsilon-greedy algorithm can achieve higher cumulative rewards than a purely greedy strategy.

Updating Reward Estimates

For each arm a, the algorithm maintains:

- N_a: The number of times arm a has been selected,
- Q_a: The current estimate of the average reward for arm a.

After observing a reward R_t from the selected arm, the estimate is updated using the incremental mean formula:

$$Q_a \leftarrow Q_a + \frac{1}{N_a}(R_t - Q_a)$$

This approach allows efficient online updates without storing the entire reward history.

Python Code Example

```python
import numpy as np

def epsilon_greedy(num_arms, num_episodes, epsilon):
    Q = np.zeros(num_arms)  # Estimated rewards
    N = np.zeros(num_arms)  # Number of times each arm is selected
    rewards = []

    for _ in range(num_episodes):
        # Exploration vs exploitation
        if np.random.rand() < epsilon:
            action = np.random.randint(num_arms)
        else:
            action = np.argmax(Q)

        # Simulate reward (for example purposes, using Bernoulli arms)
        reward = np.random.binomial(1, true_probs[action])

        # Update estimates
        N[action] += 1
        Q[action] += (reward - Q[action]) / N[action]

        rewards.append(reward)

    return Q, rewards
```

So summarizing:

- Epsilon-Greedy combines the simplicity of the greedy algorithm with a touch of randomness for better long-term performance.
- It avoids getting stuck with suboptimal arms by continuing to explore.
- The exploration rate ε needs to be carefully tuned for each problem.
- It serves as a strong baseline in many reinforcement learning applications.

Upper Confidence Bound

The Upper Confidence Bound (UCB) algorithm is one of the most widely used strategies for solving the Multi-Armed Bandit problem. It is grounded in a powerful principle known as:

"Optimism in the face of uncertainty."

Instead of relying purely on estimated rewards, UCB encourages exploration of actions that we are less certain about. In other words, the more uncertain we are about an arm, the more attractive it becomes to try it.

Intuition

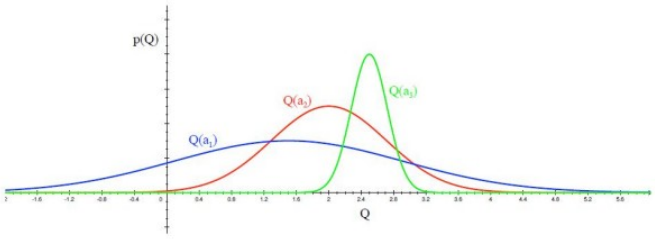

Figure: Distribution of action-value estimates for three arms. Higher variance = more uncertainty.

Imagine we have three arms a_1, a_2, a_3, each with their own distribution of estimated action-values after several rounds. Suppose the estimate for a_1 shows the highest variance — meaning we're most uncertain about it.

According to UCB, we should select a_1, not because it has the highest current average reward, but because the uncertainty around it gives it potential. If the arm is good, we benefit directly. If it's not, the act of trying it helps reduce our uncertainty, improving future decisions.

This approach leads to two beneficial outcomes:

- If our optimism is justified, we receive high rewards.
- If it's not justified, we reduce uncertainty and avoid wasting future steps on a bad arm.

The UCB1 Algorithm

UCB1 is a specific and widely-used variant of the UCB family. Here's how it works:

Steps

1. **Initialization:** Play each arm once to obtain initial reward estimates.
2. **For each time step $t \geq K$:**
   - Let $N_t(a)$ be the number of times arm a has been selected so far.
   - Compute the score for each arm a using the UCB1 formula:

$$\text{UCB}_t(a) = Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}}$$

     where:
     - $Q_t(a)$: current average reward for arm a
     - t: current time step
     - c: exploration coefficient (commonly $c = 1$)
   - Select the arm with the highest UCB value.
   - Observe the reward and update $Q_t(a)$ accordingly.

Algorithm Summary (UCB1)

```
1: Initialization: Play each of the K arms once.
2: for t = K+1 to T do
3:     for each arm a do
4:         Compute: UCB_t(a) = Q_t(a) + c * sqrt(ln(t) / N_t(a))
5:     end for
6:     Select arm a_t = argmax_a UCB_t(a)
7:     Play a_t, observe reward r_t
8:     Update mean reward Q_t(a_t)
9: end for
```

Interpreting the Formula

The UCB expression consists of two terms:

- $Q_t(a)$: encourages exploitation by choosing the arm with the highest estimated reward.
- $\sqrt{\frac{\ln t}{N_t(a)}}$: the confidence term, which encourages exploration.

The confidence term works as follows:

- Each time an arm is selected, $N_t(a)$ increases, and the uncertainty term shrinks.

$$N_t(a) \uparrow \quad \sqrt{\frac{2 \log t}{N_t(a)}} \downarrow$$

- If an arm is not selected, $N_t(a)$ remains constant, while $\ln t$ grows, increasing the uncertainty.

$$t \uparrow \quad \text{With } N_t(a) \text{ constant} \quad \sqrt{\frac{2 \log t}{N_t(a)}} \uparrow$$

- This ensures arms with high uncertainty are explored early and less often later.

Python Implementation

```
import numpy as np

def ucb1(num_arms, num_episodes, c=1):
    Q = np.zeros(num_arms)  # Average rewards
    N = np.zeros(num_arms)  # Number of times each arm was played
    rewards = []
    for t in range(1, num_episodes + 1):
        if t <= num_arms:
            action = t - 1  # Play each arm once
        else:
            ucb_values = Q + c * np.sqrt(np.log(t) / (N + 1e-5))
            action = np.argmax(ucb_values)
        reward = np.random.binomial(1, true_probs[action])  # Simulated reward
        N[action] += 1
        Q[action] += (reward - Q[action]) / N[action]
        rewards.append(reward)
    return Q, rewards
```

UCB vs Epsilon-Greedy (Quick Comparison)

Algorithm	Exploration Strategy	Adaptivity
Epsilon-Greedy	Random arm with fixed probability ε	Fixed exploration rate (unless decayed)
UCB1	Chooses arms with high reward & uncertainty	Adaptive; reduces exploration

Limitations and Considerations

- UCB assumes rewards are bounded (e.g., in [0, 1]) and independent.
- It may be overly optimistic in environments where reward distributions change over time.
- The choice of the exploration constant c affects both convergence speed and performance.

Key Takeaways

- **Multi-Armed Bandit (MAB)**: A problem of selecting actions to maximize cumulative rewards under uncertainty, balancing exploration and exploitation.
- **Reward Estimation**: The expected reward for an action is estimated by averaging past rewards.
- **Incremental Update**: Action value estimates are updated incrementally without storing all past rewards.
- **Epsilon-Greedy Algorithm**: Balances exploration (random selection) and exploitation (best-known choice) based on a parameter ε.
- **Greedy Algorithm Issues**: Tends to quickly converge to suboptimal actions due to lack of sufficient exploration.
- **Upper Confidence Bound (UCB)**: Encourages exploration of uncertain actions by adding a confidence term to the reward estimate.
- **UCB1 Algorithm**: Balances exploration and exploitation by considering both reward and uncertainty in action selection.
- **Python Implementations**: Epsilon-Greedy and UCB1 algorithms can be implemented to solve the MAB problem effectively.
- **Exploration vs. Exploitation**: The key challenge is finding the right balance between trying new actions and sticking with the best-known ones.