1 Introduction to the Multi-Armed Bandit Problem

Imagine you are faced with several slot machines (often called "one armed bandits"), each with a different, unknown probability of paying out. You have a limited number of pulls, and you want to maximise your total reward. This is the classic **k-armed bandit problem**. Formally, at each time step t you choose an action A_t from a set of k actions. After taking action a, you receive a reward R_t drawn from a stationary probability distribution that depends only on the chosen action. The goal is to maximise the expected total reward over some time horizon.

Let the **true value** of action a be

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a].$$

If you knew $q_*(a)$, you would always select the action with the highest value. In practice, you must estimate these values from experience. Denote the estimate of action a at time t by $Q_t(a)$. The simplest way to estimate $q_*(a)$ is to average all the rewards obtained after selecting a up to time t:

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

2 The Exploration–Exploitation Dilemma

At any time step, you have a choice between:

- **Exploit:** select the action that currently has the highest estimated value (the *greedy* action). This maximises immediate reward.
- **Explore:** select a different action to improve your estimate of its value, potentially discovering a better long-term choice.

If you never explore, you may miss a truly optimal action that initially appears poor due to unlucky samples. If you always explore, you fail to capitalise on the knowledge you have already gained. This conflict is the **exploration–exploitation dilemma**. It is a central challenge in reinforcement learning and appears in many forms beyond bandits.

2.1 ε-Greedy Action Selection

A simple way to balance exploration and exploitation is the **ε-greedy** method. With probability $1 - \varepsilon$ you select the greedy action; with probability ε you select uniformly at random among all actions. This ensures that all actions are sampled infinitely often as the number of steps increases, guaranteeing that the estimates $Q_t(a)$ converge to the true values $q_*(a)$. The parameter ε controls the amount of exploration.

2.2 A Simple Experiment: The 10-Armed Testbed

To illustrate the trade-off, the **10-armed testbed** was introduced. For each of 2000 randomly generated bandit problems, the true action values $q_*(a)$ were drawn from a

normal distribution with mean 0 and variance 1. Rewards were then drawn from a normal distribution with mean $q_*(A_t)$ and variance 1. A greedy method ($\varepsilon = 0$) often gets stuck with suboptimal actions, while ε-greedy methods with $\varepsilon = 0.1$ or 0.01 find the optimal action more reliably, albeit with a small performance penalty due to random exploration.

3 Incremental Update Methods for Action-Value Estimates

Storing all rewards and recomputing the average each time an action is selected would be inefficient and memory-intensive. Instead, we can derive an **incremental update rule**.

Let Q_n be the estimate of an action's value after it has been selected $n-1$ times, and let R_n be the n-th reward received for that action. The new average after n selections is:

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) = \frac{1}{n}\left(R_n + (n-1)Q_n\right) = Q_n + \frac{1}{n}\big[R_n - Q_n\big].$$

This can be written in a general form:

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\,[\text{Target} - \text{OldEstimate}].$$

In the sample-average method, the step size is $1/n$.

3.1 Nonstationary Problems

If the reward probabilities change over time, it is desirable to give more weight to recent rewards. A constant step-size parameter $\alpha \in (0, 1]$ achieves this:

$$Q_{n+1} = Q_n + \alpha\big[R_n - Q_n\big].$$

Expanding this recurrence shows that Q_{n+1} is an exponentially weighted average of past rewards and the initial estimate Q_1:

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.$$

This is called an **exponential recency-weighted average**. Because the sum of the weights is 1, it is a proper weighted average. The constant α determines how quickly old data are forgotten. For convergence in stationary problems, the step-size sequence must satisfy $\sum_{n=1}^{\infty} \alpha_n(a) = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$. The constant step size does not meet the second condition, which is why it never fully converges but can track changes in nonstationary environments.

4 Optimistic Initial Values

All the methods discussed so far depend on the initial action-value estimates $Q_1(a)$. For sample-average methods, this bias disappears once every action has been selected at least once. For constant step-size methods, the bias is permanent but decays over time.

Optimistic initialisation is a simple trick to encourage exploration. Instead of setting initial estimates to zero, set them to a value that is *optimistic* ; much higher than any reward that can actually be obtained. For example, in the 10-armed testbed where true values are around zero, setting $Q_1(a) = +5$ for all a makes the agent initially "disappointed" because the actual rewards are lower. Consequently, the agent will try other actions, exploring widely even if it follows a greedy policy.

An optimistic greedy method initially performs worse because it explores, but eventually outperforms a realistic ε-greedy method. The downside is that optimistic initialisation is only a temporary exploration boost; in nonstationary problems where the optimal action changes over time, it does not help with renewed exploration needs.

5 Upper Confidence Bound (UCB) Action Selection

The ε-greedy method explores indiscriminately; all non-greedy actions are tried with equal probability, regardless of how close their estimates are to the greedy action or how uncertain those estimates are. A more sophisticated approach is to select actions according to both their estimated value and a measure of uncertainty. The **Upper Confidence Bound (UCB)** algorithm does exactly this.

At time t, UCB selects the action that maximises

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

where $N_t(a)$ is the number of times action a has been selected before time t, and $c > 0$ controls the degree of exploration. The square-root term is an estimate of the uncertainty (or variance) of $Q_t(a)$. The logarithm ensures that the exploration bonus grows slowly over time, but it never stops growing, so all actions will eventually be tried.

5.1 Intuition Behind UCB

The idea is to consider an *upper bound* on the possible true value of each action. Actions that have been tried fewer times have larger uncertainty and thus a higher upper bound. As an action is selected more often, its $N_t(a)$ increases, reducing the bonus term. Meanwhile, actions that have not been tried for a while see their bonus increase because t grows while $N_t(a)$ stays the same. This naturally focuses exploration on actions that are either promising (high $Q_t(a)$) or highly uncertain (small $N_t(a)$).

5.2 Advantages and Limitations

UCB often performs better than ε-greedy on the 10-armed testbed, especially after the initial steps. However, UCB is more difficult to extend to general reinforcement learning settings beyond bandits. It requires careful handling of nonstationarity and is not easily combined with function approximation. However, it is a powerful principle for balancing exploration and exploitation in simple bandit problems.