Faculté
des **sciences économiques** et de **gestion**
Université de Strasbourg

Reinforcement Learning

UE 2 Machine learning

Nattirat Mayer

Fall-Winter Semester 2025-26

Chapter 2. Multi-Armed Bandits

Here comes a new chapter!

2. Multi-armed bandits

Vocabulary

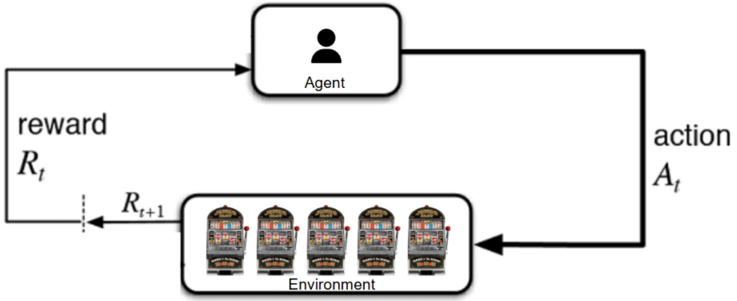
- One-armed bandit= One slot machine
- Multi-armed bandit = Multiple slot machine

Question: Playing which slot machine has the highest reward?

Not just in casinos and slot machines... but this chapter is about decision making when we have incomplete information.

Other examples of arms

- For movie recommendations, arms = movies
- For clinical trails, arms = types of treatment
- For advertisement, arms = ads
- For e-commerce, arms = products to recommend



- The loop does as $A_t, R_t, A_{t+1}, R_{t+1}, A_{t+2}, R_{t+2}, \ldots$
- The state S_t does not change
- Agent's objective: maximize cumulative rewards
- Action a = arm (one slot machine)
- Time step t = play number

- ”RL uses training information that evaluates the actions taken rather than instructs by giving the correct actions”, this is why RL needs active exploration, and evaluation of the different alternate actions and their consequences, in order to find out which action is good ”empirically”.
- Pure evaluative feedback used by RL, does not allow to assess whether it was the best ”absolute” action or not.
- Pure instructive feedback indicates the best action to be chosen. This occurs in optimization, microeconomics and game theory which derive optimal solutions or best response functions.
- This chapter presents a first learning problem: a multi-armed bandit model, which has wide applications in finance, online advertising, health care, etc.

2.1 A *k*-armed bandit problem

- The problem described:
  - Repeated choice of an action between k different options. $A_t = a \in \mathcal{A} = \{1, \ldots, k\}$ The choice is repeated T times (time steps). Here $T = 1000$
  - After each action we receive a reward r_t drawn from a stationary pdf $f(r; A_t)$ which is **unknown** to the agent

- Recall value function $v_\pi(s)$ from last session.
  This lecture we introduce $Q(a)$
- Each of the k action A_t has an expected (mean) reward:

$$q_*(a) = \mathbb{E}[R_t | A_t = a] = \int_{-\infty}^{+\infty} rf(r;a)dr$$

- If $q_*(a)$ was known, then it would be easy to choose a to maximize it
- The objective is to maximize the reward $q_*(a)$. The problem is that we do not know $q_*(a)$.

2.2 Action-value methods

- Let $Q_t(a)$ denote the estimated value of action a at step t. We define,

$$Q_t(a) \equiv \frac{\text{sum of rewards obtained at time } t-1 \text{ for } A_s = a}{\text{number of times } A_s = a \text{ has been chosen at time } t-1}$$
$$= \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

where

$$\mathbb{1}_{A_i=a} \begin{cases} 1, & \text{if action } a \text{ was chosen at time } i, \\ 0, & \text{otherwise.} \end{cases}$$

This is also called *the sample-average method for estimating action values*

- Example of how to compute $Q_t(a)$

- By the law of large numbers, if t increases to ∞

$$Q_t(a) \xrightarrow{p} q_*(a)$$

It can also be shown that (under uniform convergence)

$$\arg\max_a Q_t(a) \xrightarrow{p} \arg\max_a q_*(a)$$

- 2.2.1 A greedy action is to always exploit and is defined by

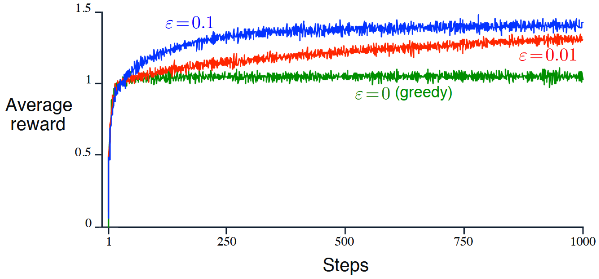
$$A_t \equiv \arg\max_a Q_t(a)$$

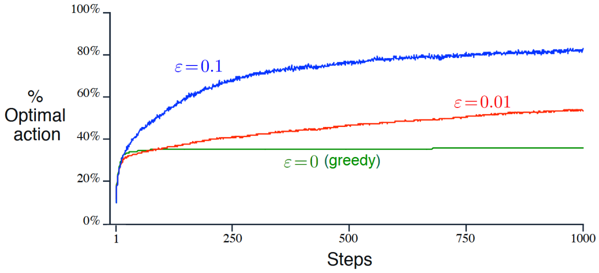
  - A greedy strategy makes sense for maximizing $q_*(a)$. For small t, however, it is a poor choice as it is likely that $Q_t(a)$ be very different from $q_*(a)$.

- Exploration vs. Exploitation Dilemma
  - **Exploration** = finds more information about the environment
  - **Exploitation** = exploits known information to maximize reward
  - The goal is to strike a balance between exploration and exploitation
- We introduce ϵ-greedy strategy
  - where ϵ is the degree of **randomness**
  - In general, $\epsilon = 0.1$ means we explore 10% of the time or $\epsilon = 0.01$ means we explore 1% of the time

Greedy Vs ϵ-greedy in action

Figure 2.2. Average performance of ϵ-greedy action-value methods on the 10-armed testbed





Greedy Vs ϵ-greedy

- In greedy strategy, we always exploit
- In ϵ-greedy strategy, we are usually greedy, but with probability ϵ that we instead pick an action at random (possibly the greedy action again). I.e., we re-sample other actions with **uniform probability**

However, how can we improve ϵ-greedy strategy ?

Before we improve how we improve the ϵ-greedy strategy, we first look at how we compute $Q_t(a)$. Previously, we got

$$\begin{aligned} Q_t(a) &\equiv \frac{\text{sum of rewards obtained at time } t-1 \text{ for } A_s = a}{\text{number of times } A_s = a \text{ has been chosen at time } t-1} \\ &= \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \end{aligned}$$

Now, we concentrate on a single action. Let Q_n denote the estimate of the action value after it has been selected $n-1$ times. This can be written as the sample average,

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}.$$

Note that n is the number of time action a has been selected.

How can we do this incrementally (without storing all the rewards)?

2.4 Incremental implementation

Derivation of incremental update

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

Or in words,
Newestimate $\leftarrow$ Oldestimate + Stepsize $\big[$ Target - Oldestimate $\big]$

A simple bandit algorithm

```
Initialize, for $a = 1$ to $k$:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
Loop for $t = 1, \dots, 1000$:
    $A \leftarrow \begin{cases} \arg\max_a Q(a), & \text{with probability } 1-\varepsilon, \\ \text{a random (uniform) value in } \{1, \dots, k\}, & \text{with probability } \varepsilon. \end{cases}$
    $R \leftarrow$ a random draw from $f(r; A)$
    $N(A) \leftarrow N(A) + 1$
    $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big(R(A) - Q(A)\big)$
```

- Exercise. Convert the algorithm into a computer code for R or Python.

- Exercise: incremental update in action

Pull	1	2	3	$Q(a_1,a_2,a_3)$
	$Q_1(a_1) = 0$	$Q_1(a_2) = 0$	$Q_1(a_3) = 0$	$Q(0,0,0)$
1	$R_1(a_1) = +5$	-	-	
2	-	$R_2(a_2) = +8$	-	
3	-	-	$R_3(a_3) = +3$	
4	-	$R_4(a_2) = +7$	-	
5	-	$R_5(a_2) = +9$	-	
6	-	$R_6(a_2) = +2$	-	

2.5 Tracking a nonstationary problem

- What happens if the reward probability distributions change over time?
  In this case, simple sample averages are not a good idea.
  A better approach is the **exponential, recency-weighted average**:

$$\begin{aligned} Q_{n+1} &= Q_n + \alpha\left[R_n - Q_n\right] \\ &= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i, \end{aligned}$$

  where $\alpha \in (0, 1]$ is a constant step-size parameter.

So far...

- We improve the efficiency of how we compute $Q(a)$ with incremental update
- Still one question remains, how can we improve ϵ-greedy strategy ? (i.e., how can we explore in a better way?)

2.6 Optimistic Initial Values

- Suppose we initialize the action values optimistically,

Pull	1	2	3	$Q(a_1,a_2,a_3)$
1	$Q_1(a_1) = +20$	$Q_1(a_2) = +20$	$Q_1(a_3) = +20$	Q(+20,+20,+20)
2	$R_2(a_1) = +5$	-	-	Q(+5,+20,+20)
3	-	$R_3(a_2) = +8$	-	Q(+5,+8,+20)
4	-	-	$R_4(a_3) = +3$	Q(+5,+8,+3)
5	-	$R_5(a_2) = +7$	-	Q(+5,+7.5,+3)
6	-	$R_6(a_2) = +9$	-	Q(+5,+7,+3)

- Sutton and Barto (2018) initialize

$$Q_1(a) = 5 \quad \text{for all } a$$

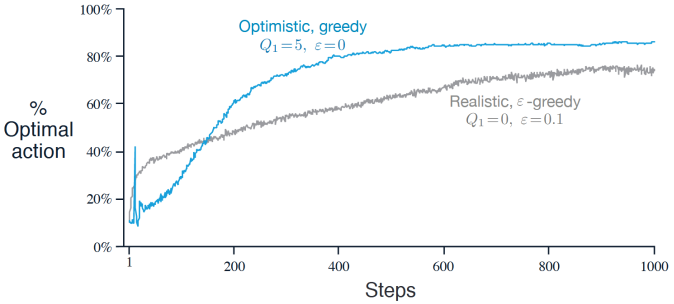


Figure 2.3: The effect of optimistic initial action-value estimates on the 10-armed testbed

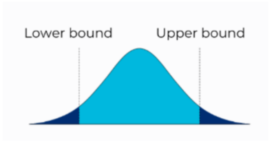
Optimistic Initial Values

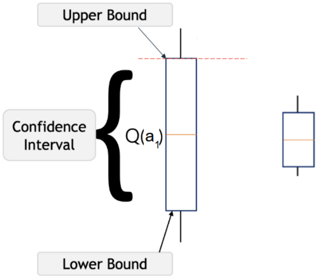
- Encourage exploration, but only temporarily — eventually the agent exploits.
- Not suitable for **nonstationary** problems (when reward probability distributions change over time).

Are there more clever strategies for exploration?

2.7 Upper confidence-bound action selection

- Recall, Confidence Interval (CI)

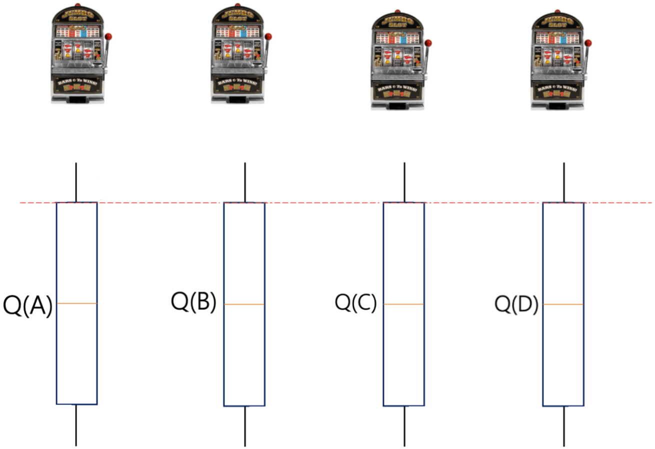




- Wide CI $\rightarrow$ the estimate is very uncertain (we don't know much yet).
- Narrow CI $\rightarrow$ the estimate is more precise (we have stronger evidence).

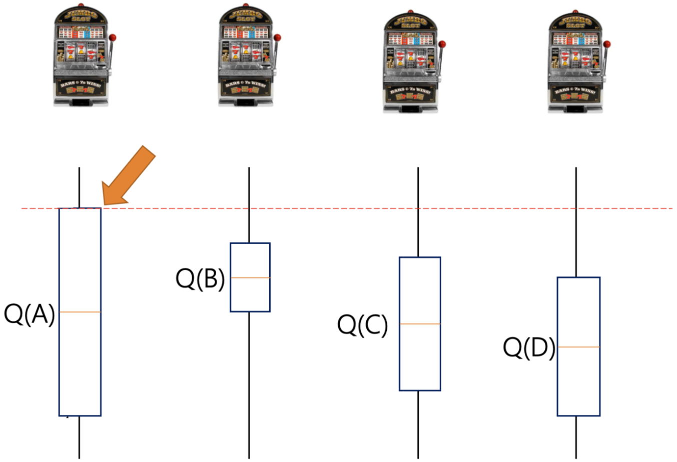
How UBC algorithm works...

- Initialize all the arms
  (wide CI because we do not know (uncertain) about the arms)



Reference: GeeksforGeeks, 2020

- The algorithm chooses the arm with the highest upper confidence bound. As time goes on, the uncertainty bonus shrinks for frequently tried arms.



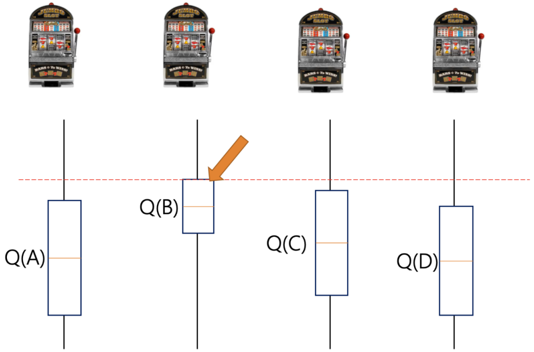
Reference: GeeksforGeeks, 2020

- The algorithm will **mostly exploit** the best arm but the $\ln t$ term, it will still occasionally test other arms to avoid missing a better one.
- A UCB-action selects the optimal action according to

$$A_t \equiv \arg\max_a Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}$$

  for some tuning parameter c to be chosen in function of the data.
  - The constant $c \geq 0$ controls the degree of exploration:
    - $c = 0$ means no exploration.
  - The square-root term is a measure of uncertainty.
  - We maximize over the upper bound of the possible true action values $Q_t(a)$.

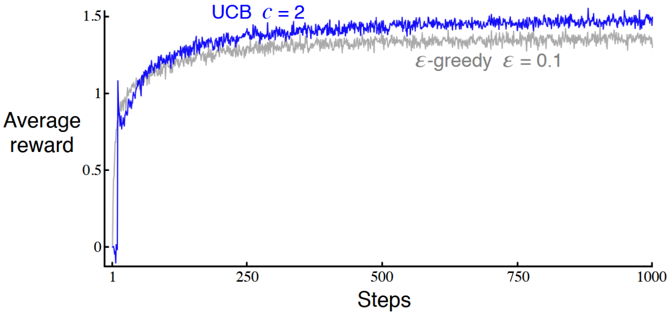
- Eventually, the algorithm spends almost all time on the optimal arm (exploitation).



Reference: GeeksforGeeks, 2020

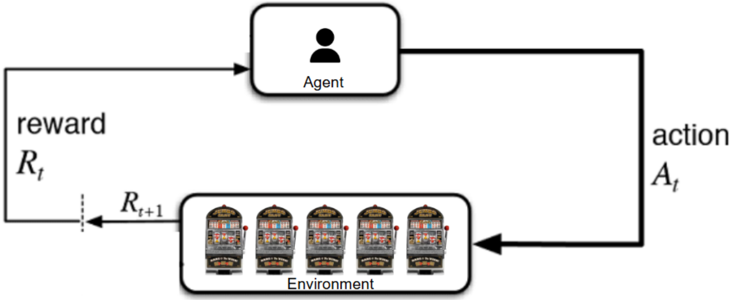
- UCB Vs ϵ-Greedy Algorithm

Figure 2.4. Average performance of UCB action selection



- Interesting complementary information on UCB is provided by Lilian Weng at: https://github.com/lilianweng/multi-armed-bandit/tree/master
- Exercise:
  - a theoretical guide on how to best calibrate the constant term c (you can for instance use the material provided by Lilian Weng)
  - illustrate how the choice of c influence the empirical performance of your algorithm (represent a figure like Fig. 2.4)

So far...



- In the standard multi-armed bandit:
  - We sit at a casino and pull the levers
  - The state S_t does not change.
- **What happens if the state *does* change?**

2.9 Contextual Bandits (Associative Search)

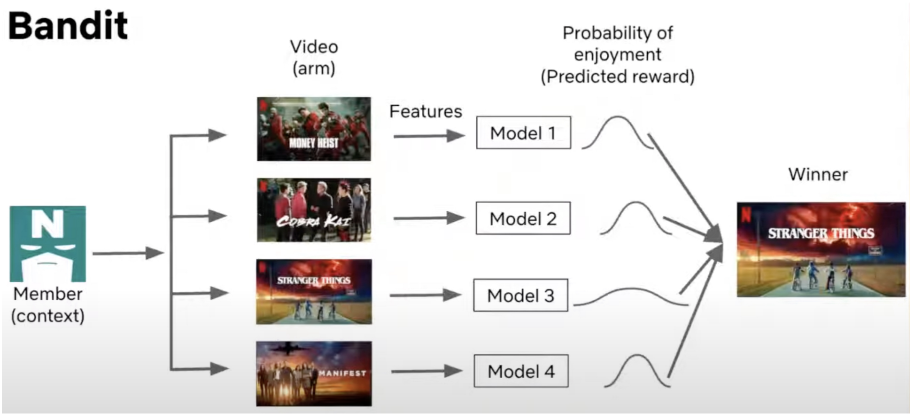
- In contextual bandits, the choice of action depends on the observed state (context):

$$A_t = \arg\max_a q_*(a \mid S_t), \quad S_t = s(S_{t-1}, \ldots)$$

- By contrast, in the full RL problem, the next state also depends on the action:

$$A_t = \arg\max_a q_*(a \mid S_t), \quad S_t = s(S_{t-1}, A_{t-1}, \ldots)$$

$$\text{User's context} = \begin{bmatrix} \text{Age} \\ \text{Location} \\ \text{Watch History} \\ \text{Time of Day} \\ \vdots \end{bmatrix}$$



Reference: vinija.ai

2.10 Other Multi-Armed Bandit Algorithms

For further research and study, some additional bandit algorithms include

- **Thompson Sampling** – Bayesian approach using posterior sampling.
- **Gradient Bandit Algorithms** – Learn preferences with softmax action selection.
- **EXP3** – Handles adversarial bandit problems.
- **KL-UCB** – Uses KL divergence for sharper confidence bounds.
- **Contextual Bandits** – Incorporates side information (e.g. user profile).

2.10 Summary of Chapter 2

- What have we learned?
  - Action-value Q(a) which comes in different notations
    - $q_*(a)$ = mean reward we do not know
    - Calculate $Q_t(a)$ by sample-average method but not so efficient
    - Therefore, we use incremental implementation of Q_{n+1}
  - Various algorithms for **exploration-exploitation trade-off**
    - Greedy
    - ϵ-greedy
    - Optimistic initial value
    - UBC
    - Contextual Bandit

Wrap-up and Next Steps

- **Exercise:** Solve Exercise 2.11 (p.44) in the textbook.
- **Up next: Chapter 3 – Markov Decision Processes (MDPs)**
  - Bandits: no states, only choose actions $\rightarrow$ short-sighted.
  - MDPs: introduce *state transitions*, delayed rewards, and planning.
  - Foundation of most RL applications (games, robotics, etc.).
- Get ready: MDPs are the bridge from simple slot machines to full-fledged intelligent agents
- Start thinking about your project, decide whether you want to work individually or in pairs. If in pairs, try to find your partner now.