Multi-arm Bandits

Action-Value Methods

- For an agent to decide which action yields the maximum reward, concept of probability is used to define these values using the action-value function.
- The value of selecting an action is defined as the **expected** reward received when taking that action from a set of all possible actions.
- Since the value of selecting an action is not known to the agent, so we use the 'sample-average' method to **estimate** the value of taking an action.

The estimated value on the *t*-th time step as $Q_t(a)$.

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)}.$$

The simplest action selection rule is to select the action (or one of the actions) with highest estimated action value, that is, to select at step t one of the greedy actions, A^*_t for which $Q_t(A^*_t) = \max_a Q_t(a)$.

Bandit problem

- K-armed bandit problem, so named by analogy to a slot machine.
- One-armed bandit, has one lever.
- Through repeated action selections (say, 1000 times) you are to maximize your winnings by concentrating your actions on the best levers.
- ***Regret*** is difference between optimal selection of levels (best outcome) and your choice, may be nonbest outcome.

Decision-making problem

- The multi-armed bandit (MAB) problem is a decision-making problem where you choose between multiple options, each with an unknown payoff (providing a reward drawn from an unknown probability distribution).
- The goal is to maximize your rewards over time.
- The challenge lies in choosing the best arm to pull, balancing the need to explore different arms to learn about their reward distributions and exploiting the known arms that have provided high rewards.
- **Formal Representation**
- **Arms**: K independent arms, each with an unknown reward distribution.
- **Rewards**: Each arm *i* provides a reward R_i, drawn from an unknown distribution with an expected value μ_i.
- **Objective**: Maximize the cumulative reward over T trials.

Naïve Method

- In a multi-armed bandit problem, an agent chooses between *K* different actions and receives a reward based on the chosen action.
- For selecting an action by an agent, we assume that each action has a separate *distribution of rewards* and there is at least one action that generates maximum numerical reward.
- Thus, the probability distribution of the rewards corresponding to each action is **different** and is unknown to the agent (decision-maker).
- The goal of the agent is to identify which action to choose to get the maximum reward (best distribution) after a given set of trials.

I randomly choose each for exploration. Say, 250*(0.7+0.3+0.55+0.4)=488 wins

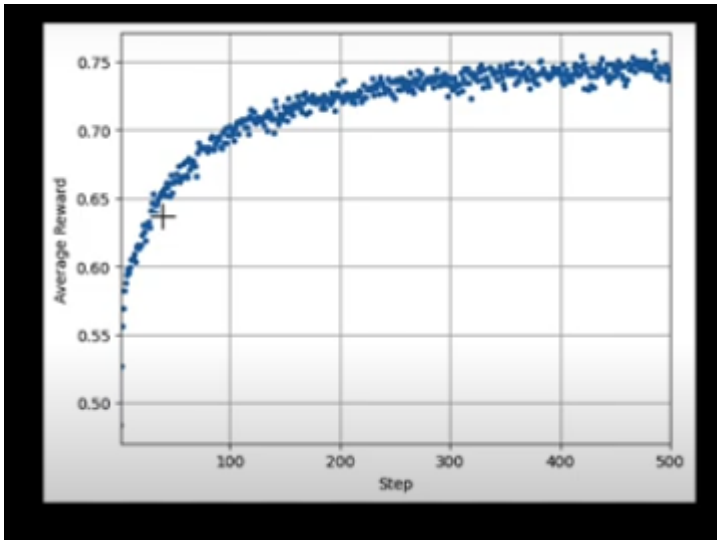
Outcome is **unknown to the agent**. I choose second, with an hope to get back more and say 100 times win but for 1000 times not, so totally exploitation is not good.

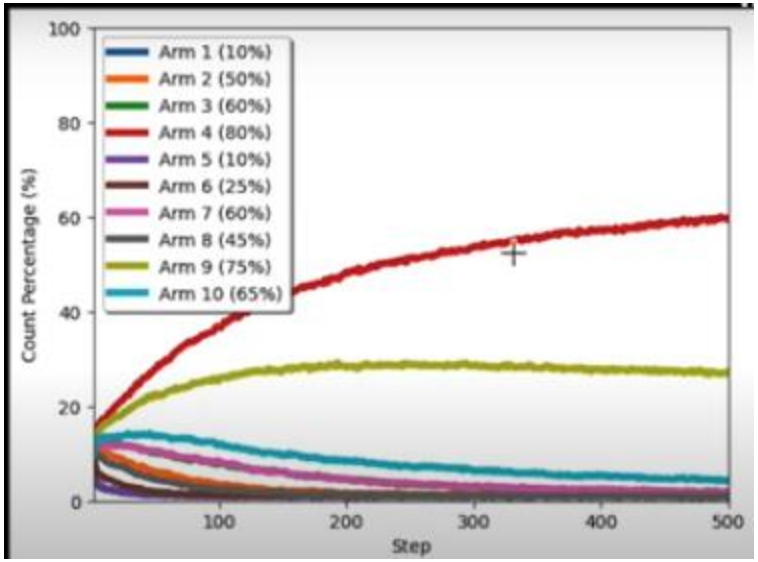
Machine Played	Won/Lost
A	1
B	0
A	0
C	1
D	0
C	1

This process continues and is repeated until the agent maximizes its reward.

Conflict

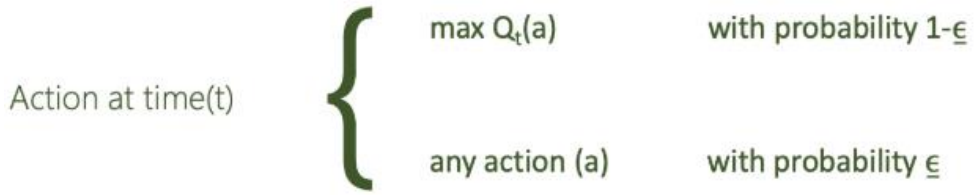
- *Exploration* allows an agent to improve its current knowledge about each action, hopefully leading to **long-term benefit**.
- Improving the accuracy of the estimated action-values, enables an agent to make more informed decisions in the future.
- *Exploitation* on the other hand, chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates.
- But by being greedy with respect to action-value estimates, may not actually get the most reward and lead to sub-optimal behaviour.
- An agent cannot, however, choose to do both simultaneously, which is also called the exploration-exploitation dilemma.


Average Reward
0.75
0.70
0.65
0.60
0.55
0.50
100
200
300
400
500
Step


Arm 1 (10%)
Arm 2 (50%)
Arm 3 (60%)
Arm 4 (80%)
Arm 5 (10%)
Arm 6 (25%)
Arm 7 (60%)
Arm 8 (45%)
Arm 9 (75%)
Arm 10 (65%)
Count Percentage (%)
100
80
60
40
20
0
100
200
300
400
500
Step

Epsilon-Greedy Method

- Epsilon-Greedy is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly.
- The epsilon-greedy, where epsilon refers to the probability of choosing to explore (randomly picking an action), exploits most of the time with a small chance of exploring.




```
p = random()

if p < ε:
   pull random action
else:
   pull current-best action
```

Algorithm of Epsilon-Greedy

Initialize the estimated values of all arms to zero or a small positive number.

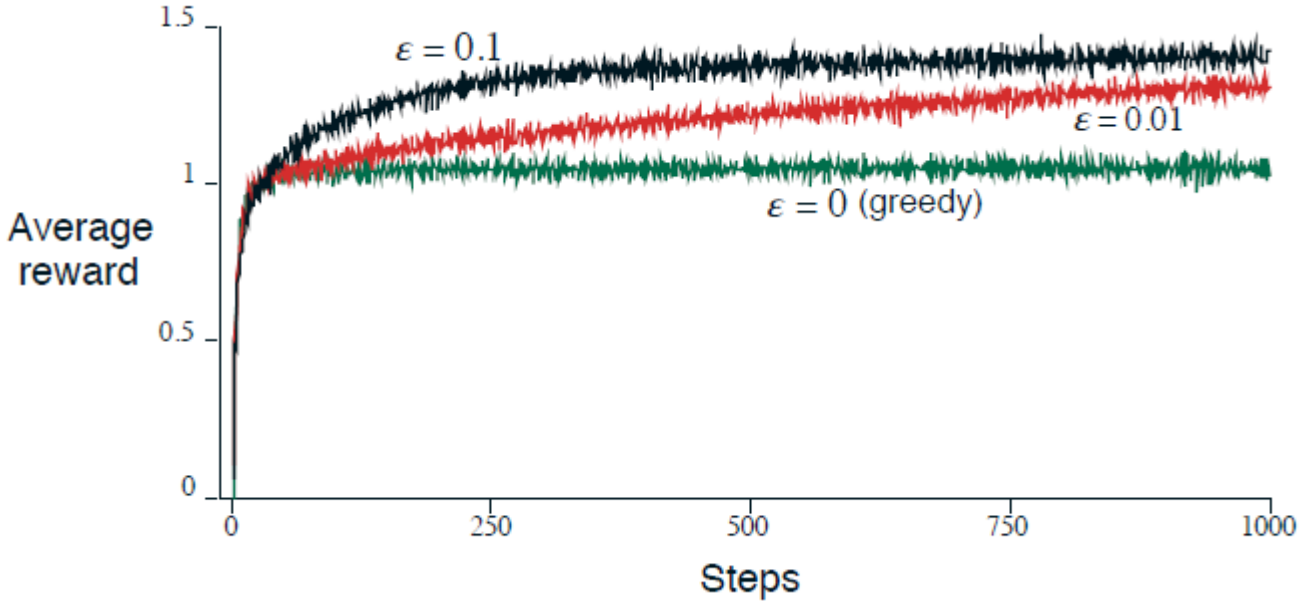
For each trial:

Generate a random number between 0 and 1.

If the number is less than ϵ, select a random arm (exploration).

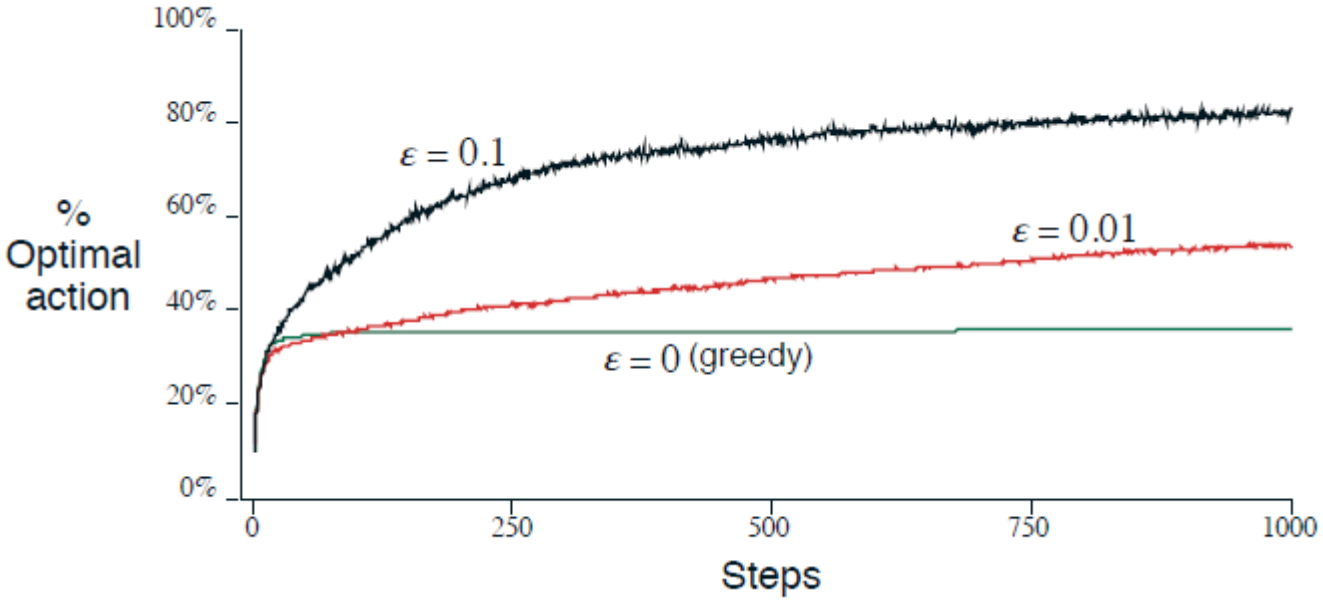
Otherwise, select the arm with the highest estimated reward (exploitation).

Update the estimated reward of the selected arm based on the observed reward.



Averages over 2000 tasks, considering action-value estimates.

The greedy method performs significantly worse in the long run because it often gets stuck performing suboptimal actions.



The greedy method found the optimal action in only approximately one-third of the tasks.

The epsilon-greedy methods eventually perform better because they continue to explore, and to improve their chances of recognizing the optimal action. The ε = 0.1 method explores more, and usually finds the optimal action earlier, but never selects it more than 91% of the time.

The ε = 0.01 method improves more **slowly**, but eventually performs better than the ε = 0.1 method on both performance measures.

Regret Minimization

- The action you regret the most is the one that should have been (more likely) used or taken.
- So the probability of taking this action is proportional to how deep you regret you haven’t taken it.
- The regret is expressed as the difference between the payoff of a possible action and the payoff of the action that has been actually taken.
- Payoff function as ***u***: ***regret = u(possible action) – u(action taken)***
- Regret minimization refers to the concept of an agent actively trying to make decisions that minimize the difference between the rewards it actually receives and the rewards it could have received by choosing the optimal action at each step.
- *Comparison to optimal policy and Focus on long-term performance*

Solutions

- Asymptotic correctness:

  Giving guarantee that eventually you will be selecting the arm which has the highest payoff where $T \rightarrow \infty$

- Regret Optimality

  Suppose, from the beginning (T=0), I know which is the best arm to pull and so keep pulling the arm over and over again.

  What will be my expected payoff?

- PAC optimality (Probably Approximately Correct)

  The PAC theory is an answer to finding the relationship between the true error rate and the number of training samples.

  In PAC RL, the goal is to find a policy that's close to optimal (ε-optimal) with a certain probability.

- Lower bound is logT : Regret not fall below logT (T time steps)

Introduction to Upper Confidence Bound (UCB)

- Employing a strategy that balances exploration and exploitation through a confidence interval-based approach.
- This tradeoff involves balancing the need to explore new actions to discover their potential rewards (exploration) with the need to exploit known actions that yield high rewards (exploitation).
- The core idea is to select actions based not only on their estimated rewards but also on the uncertainty or confidence in these estimates.
- Let t = current time step
  *a*_t is the decision or action taken by the agent at time step t
- y_t is the reward or outcome at time step t

For instance, *a*_1 means action taken at time step *t*=1
y_1 is the reward or outcome at time step t =1

policy π: [(a_1, y_1), (a_2, y_2)…….(a_(t-1),y_(t-1))] -> a_t

Algorithm

- **Initialization**: Initialize the count of pulls and the estimated rewards for each arm. *Initially, each arm is pulled once to gather preliminary data.*
- **Selection**: At each step, select the arm that maximizes the UCB value, which is a combination of the estimated reward and a term that represents the uncertainty or confidence interval.
- **Update**: After selecting an arm and receiving the reward, update the count of pulls and the estimated reward for that arm.
- Mathematically, the UCB value for arm at time t

$$UCB_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{2 \log t}{n_i(t)}}$$

where:

- $\hat{\mu}_i(t)$ is the estimated mean reward of arm i at time t.
- $n_i(t)$ is the number of times arm i has been pulled up to time t.
- t is the current time step.

Terminology

$$UCB_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{2\log t}{n_i(t)}}$$

- Estimated Reward for arm *i* is the average of the rewards obtained up to time *t*.

$$\hat{\mu}_i(t) = \frac{1}{n_i(t)} \sum_{s=1}^{t} r_{i,s}$$

where $r_{i,s}$ is the reward received from arm *i* at the s-th pull.

- Confidence Interval: The confidence interval $\sqrt{\frac{2\log t}{n_i(t)}}$ represents the uncertainty in the reward estimate.
- It is derived from the Hoeffding inequality, which provides a bound on the probability that the estimated mean deviates from the true mean.
- The term grows logarithmically with time, ensuring that the confidence interval shrinks as more data is collected.
- The factor of 2 is a scaling constant that can be adjusted based on the desired confidence level.

Explanation

- The UCB algorithm inherently balances exploration and exploitation through its selection mechanism.
- The estimated reward encourages exploitation by favoring arms with higher known rewards.
- The confidence interval term encourages exploration by favoring arms that have been pulled fewer times, thus accounting for the uncertainty in their reward estimates.
- When an arm has been pulled many times, the confidence interval term becomes smaller, reducing the incentive to explore that arm further.
- Conversely, arms that have been pulled fewer times have larger confidence intervals, making them more attractive for exploration.
- This dynamic adjustment ensures that the algorithm explores sufficiently to gather information about all arms while exploiting the best-known arms to maximize rewards.

Advantages

- **Theoretical Guarantees**: The UCB algorithm provides strong theoretical guarantees on performance, often expressed in terms of regret bounds.

  Regret is the difference between the reward obtained by the algorithm and the reward that could have been obtained by always selecting the best arm. *UCB algorithms typically achieve logarithmic regret, which is optimal for the multi-armed bandit problem.*

- **Simplicity and Efficiency**: The UCB algorithm is relatively simple to implement and computationally efficient. The calculations for UCB values involve basic arithmetic operations, making it suitable for real-time applications.

- **Adaptability**: The UCB algorithm can be adapted to various settings, including non-stationary environments where the reward distributions change over time. Extensions such as UCB1-Tuned and Discounted UCB modify the confidence interval term to account for changing conditions.