ECE433/COS435 Spring 2024 Introducton to RL

Lecture 2: Math Review and Bandit

In this lecture, we review very basic mathematics and start with bandit as an motivating problem for RL.

1. Probability basics

1.1. Probability Space

A probability space is a mathematical construct that provides a formal model of a random experiment. It is defined as a triple $(\Omega, \mathcal{F}, P)$ where:

Ω is the sample space, representing all possible outcomes.
$\mathcal{F}$ is a σ-algebra on Ω, representing all the events.
P is a probability measure, which assigns a probability to each event in $\mathcal{F}$.

A probability space has three basic properties:
(1) $P(\Omega) = 1$
(2) $P(\emptyset) = 0$
(2) For disjoint events $A, B \in \mathcal{F}$, $P(A \cup B) = P(A) + P(B)$.

1.2. Random Variables

A random variable is a function from a sample space Ω to the real numbers $\mathbb{R}$. It assigns a real number to each outcome in the sample space.\subsection{Random Variables and Probability Distributions}
The foundation of Monte Carlo methods lies in probability theory.

Random Variable: A function mapping sample w to a real value. Also viewed as a variable whose possible values are numerical outcomes of a random phenomenon.

$$X : \Omega \to \mathbb{R}$$

where Ω is the sample space and $\mathbb{R}$ represents the set of real numbers.

Probability Distribution: Describes how probabilities are distributed over the values of the random variable. Let's focus on discrete random variable X and its probability mass function is

$$P(X = x), \quad x \in \mathbb{R}$$

1.3. Expectation and Variance

Expectation and variance for discrete random variable

$$\mathbb{E}[X] = \sum_x x P(X = x)$$

and

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

1.4. Conditional Probability

The conditional probability of an event A given that B has occurred is given by

$$P(A|B) = \frac{P(A \cap B)}{P(B)},$$

with $P(B) > 0$.

1.5. Bayes' Theorem

Bayes' Theorem is a fundamental theorem in probability theory and statistics. It describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

1.6. Independence

We say events A and B are independent if

$$P(A \cap B) = P(A)P(B).$$

Random variables X and Y are independent if for all x and y, $P(X = x \cap Y = y) = P(X = x)P(Y = y)$. If X, Y are independent,

$$\mathbb{E}[f(X)g(Y)] = \mathbb{E}[f(X)]\mathbb{E}[g(Y)]$$

2. Monte Carlo Methods

Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to obtain numerical results. They are particularly useful for simulating systems with numerous coupled degrees of freedom. The method was named after the Monte Carlo Casino in Monaco because of its reliance on randomness, akin to the randomness of casino games.

The primary use of Monte Carlo methods in computational science is for estimation. Suppose we want to estimate $\mathbb{E}[f(X)]$. We draw i.i.d. samples $X_1, \ldots, X_n$ and construct the following Monte Carlo estimate:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} f(X_i)$$

where N is the number of samples and X_i are the independent identically distributed samples drawn from the distribution of X.

By the law of large numbers, as N goes to infinity,

$$\mathbb{P}(|\hat{\mu} - \mathbb{E}[f(X)]| > \epsilon) \to 0$$

for any $\epsilon > 0$.

3. Maximum Likelihood Estimation (MLE)

Given a statistical model with parameter θ and observed data X, the likelihood $L(\theta)$ is defined as:

$$L(\theta) = P(X|\theta)$$

where $P(X|\theta)$ denotes the probability of observing the data X given the parameter θ. It's important to note that likelihood is a function of θ with X held fixed.

Maximum Likelihood Estimation (MLE) is a method used to estimate the parameters of a statistical model. It selects the parameter values that maximize the likelihood function, implying the parameters for which the observed data is most probable.

3.1. MLE Formulation

The MLE $\hat{\theta}$ is defined as:

$$\hat{\theta} = \arg\max_{\theta} L(\theta)$$

Alternatively, it's often more convenient to maximize the log-likelihood, as the logarithm is a monotonic transformation. The log-likelihood is given by:

$$\ell(\theta) = \log L(\theta)$$

Thus, the MLE can also be obtained by:

$$\hat{\theta} = \arg\max_{\theta} \ell(\theta)$$

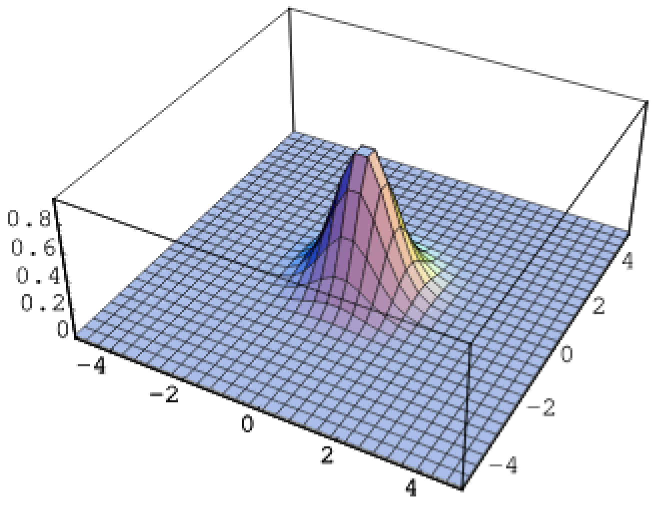
3.2. Properties of MLE

MLE has several important properties:

1. **Consistency**: As the sample size increases, the MLE converges in probability to the true parameter value.
2. **Asymptotic Normality**: Under certain conditions, the distribution of the MLE approaches a normal distribution as the sample size increases.

3.3. Estimating a Population Mean

Consider a sample $X_1, X_2, \ldots, X_n$ from a normal distribution $N(\mu, \sigma^2)$ with known σ^2 and unknown μ.



The likelihood function is:

$$L(\mu) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(X_i-\mu)^2}$$

The log-likelihood is:

$$\ell(\mu) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(X_i - \mu)^2$$

Maximizing $\ell(\mu)$ with respect to μ gives the MLE $\hat{\mu}$ as the sample mean:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

4. Gradient Optimization

Gradient Optimization is an essential technique in machine learning, deep learning, and reinforcement learning for finding minima or maxima of functions. It involves iteratively moving in the direction of the steepest descent, guided by the gradient.

Consider a function $f : \mathbb{R}^n \to \mathbb{R}$. The goal is to find a point $\mathbf{x}^*$ that minimizes $f(\mathbf{x})$.

First-Order Optimal Condition

Assume f is differential. An optimal solution x^* satisfies the **stationary condition** that

$$\nabla f(\mathbf{x}^*) = 0.$$

Gradient Descent Update Rule

The gradient descent algorithm updates the parameters $\mathbf{x}$ as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

Where:

- $\mathbf{x}_t$ is the parameter vector at iteration t.
- η is the learning rate.
- $\nabla f(\mathbf{x}_t)$ is the gradient of f at $\mathbf{x}_t$.

Practical Examples

Example 1: Linear Regression

In linear regression, the Mean Squared Error (MSE) between predicted and actual values is minimized. Given a dataset $\{(x_i, y_i)\}$, the MSE is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2$$

Gradient descent finds β_0 and β_1 that minimize this MSE.
More generally, the problem can be written in vector forms

$$\min_{\beta} MSE := \|y - X\beta\|^2$$

Gradient descent

$$\beta_{t+1} = \beta_t - \eta X^T(X\beta_t - y)$$

finds the optimal solution

$$\beta^* = (X^T X)^{-1} X^T y$$

with appropriate choice of learning rate.
For this course, you need to have basic understanding about why gradient method converges to optimal solution to linear regression. You also need to understand the regression more generally in the context of supervised learning.

Example 2: Logistic Regression

Logistic regression uses a logistic function to model a binary dependent variable. The cost function, often the cross-entropy, is:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))]$$

Where $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ is the logistic function, and gradient descent is used to minimize $J(\theta)$.
For this course, you need to understand the cross-entropy loss and the approach of logistic regression to general classification problem.

Gradient Optimization in Deep Learning

In training deep neural networks, gradient optimization is much harder due to the complex nonconvex landscope of loss function. For examples:

- **Vanishing or Exploding Gradients:** Addressed through normalized initialization, batch normalization, or activation functions.
- **Local Minima and Saddle Points:** Overcome with stochastic gradient descent and momentum.

5. Multi-Armed Bandits (MAB)

In this section, we aim to understand the basic concepts, challenges, and algorithms associated with multi-armed bandit problems in the context of reinforcement learning.

A multi-armed bandit (MAB) problem is a model for understanding decision-making in uncertain environments. It is named after a slot machine (or "one-armed bandit") with multiple arms, where each arm represents a different action, and the payouts of each arm are unknown to the player.

MAB is a simplified model of reinforcement learning, where the key challenge is to explore different actions, learning their expected payoffs and the optimal action. MAB is much simpler than RL because any chosen action only affects the immediate payoff but would not affect future of the game.

5.1. Definition

Consider a set of N slot machines, also known as bandits. Each bandit has a different probability distribution of rewards, which are unknown to the player. The player must decide which bandit to play over a series of trials to maximize their cumulative reward.

Let A be the number of arms (bandits). Denote each arm by a, where $a = 1, 2, \ldots, N$. Each arm a would yield a random reward in $\{0, 1\}$ with an unknown reward distribution and unknown mean μ_a. Let $R_{a,t}$ be the reward received from playing arm a at time t, which is a Bernoulli random variable. At each time step t, the player selects an arm a_t to play based on a strategy or policy. This decision is typically influenced by the history of played arms and received rewards up to time t.

The goal is to maximize the cumulative reward over a given number of plays (time steps), T. Formally, this is:

$$\max \mathbb{E}\left[\sum_{t=1}^{T} R_{a_t,t}\right] \tag{1}$$

where a_t is the arm played at time t.

Discussion: Precisely speaking, what is the variable to maximize in the above optimization problem?

We seek an algorithm that takes as input all prior knowledges and observations including previous actions and outcomes and maps the input to actions. The algorithm is adaptive and online in nature.

Expected Reward and Optimal Arm

- The expected reward of arm a is denoted as $\mu_a = \mathbb{E}[R_a]$.
- Total expected reward after T trials is $\sum_{t=1}^{T} \mu_{a_t}$.
- The optimal arm a^* has the highest expected reward, $\mu^* = \max_a \mu_a$.

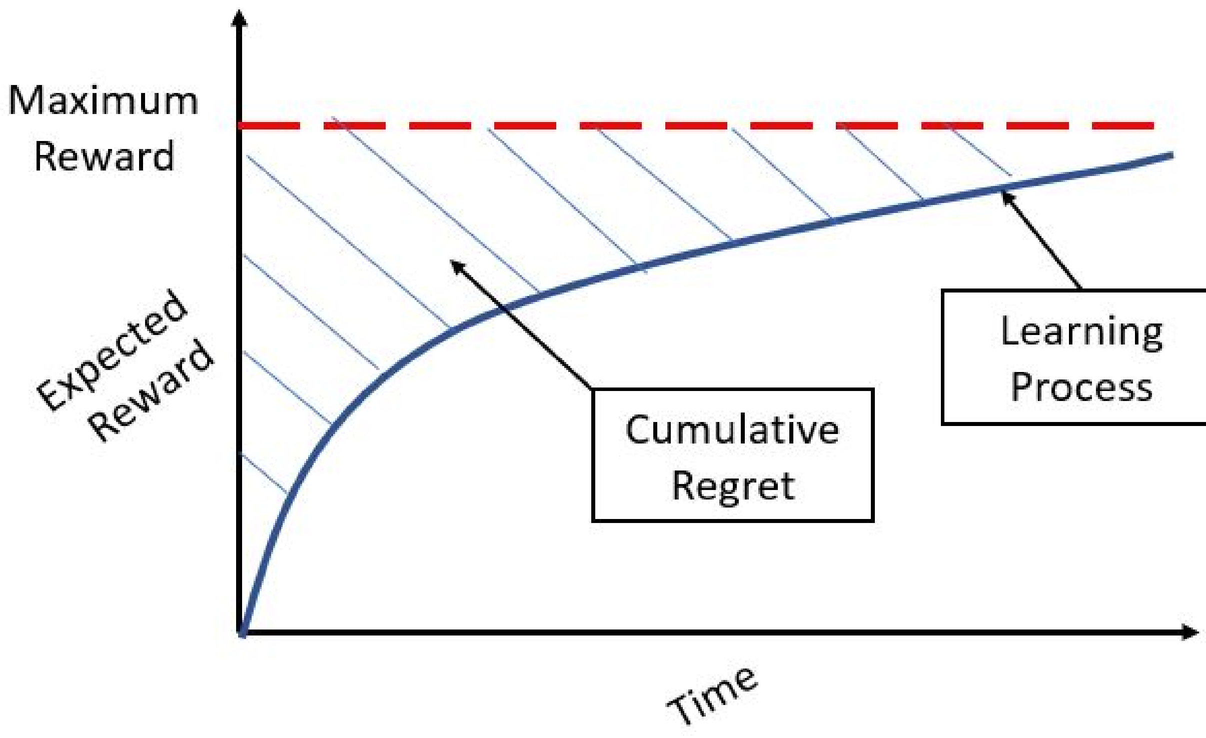
Regret

Regret is a measure of the performance of a strategy and is defined as the difference between the reward that would have been obtained by always playing the best arm and the reward actually obtained.

The simple regret measures the optimality gap at a time step:

$$\text{Simple Regret}(t) = \mu^* - \mu_{A_t} \tag{2}$$

Simple regret is always non-negative

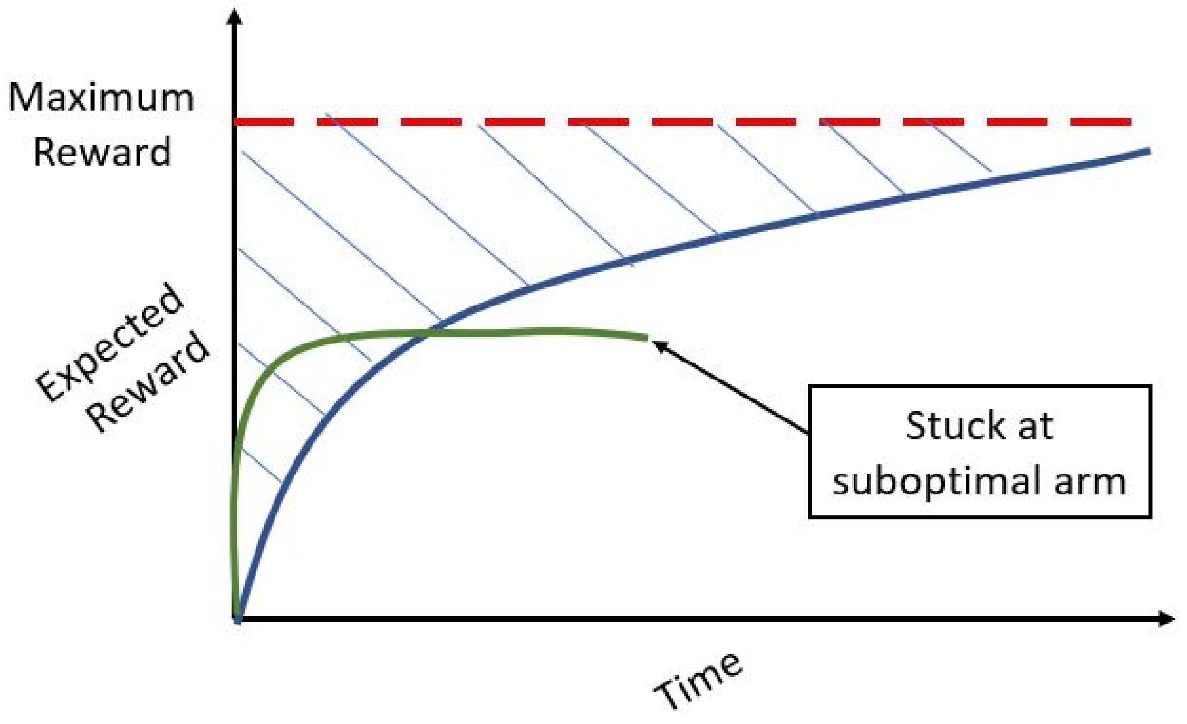


Mathematically, the cumulative regret after T trials is:

$$\text{Regret}(T) = T \cdot \mu^* - \sum_{t=1}^{T} \mu_{A_t} \tag{3}$$

The cumulative regret is non-negative and monotonically increasing.

If the learning process doesn't go well, an algorithm could stuck at a suboptimal arm:

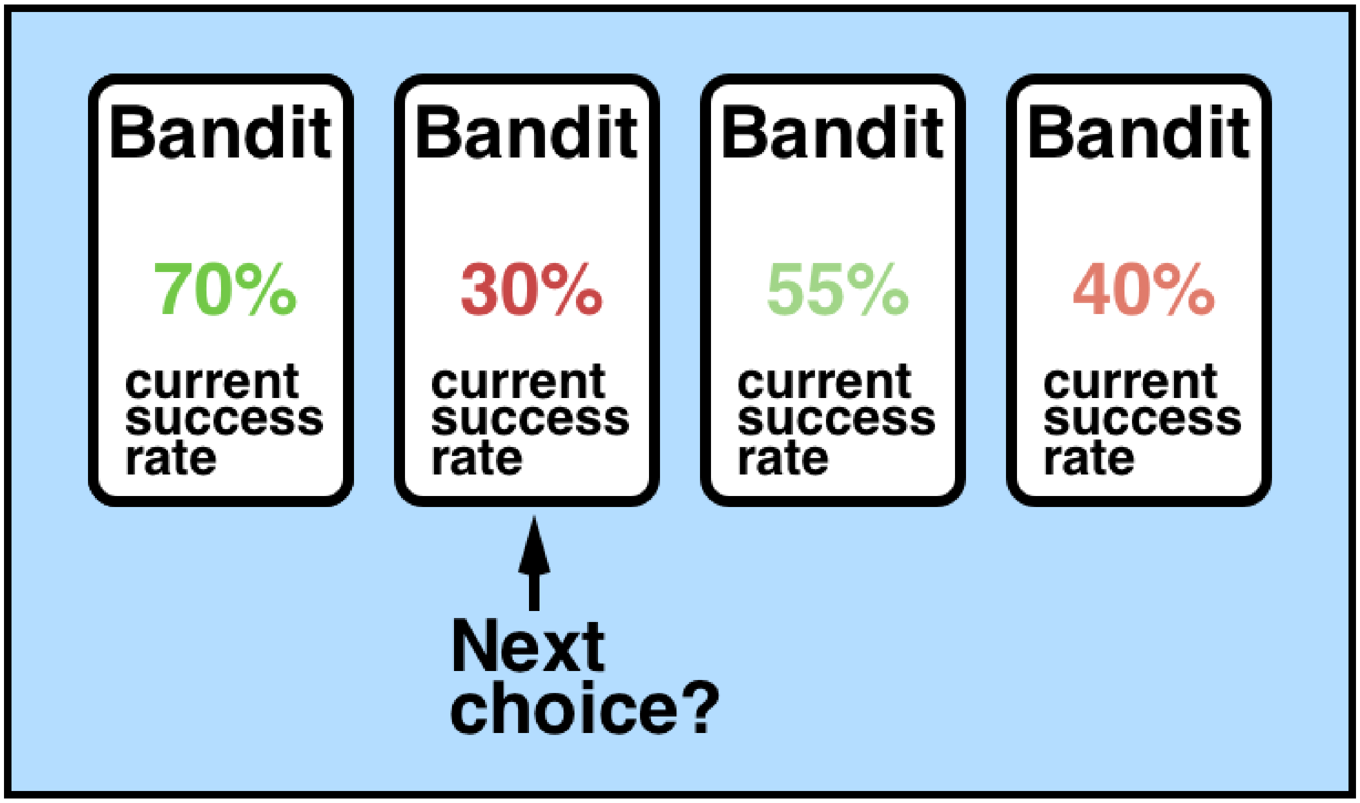


In this case, the simple regret would not reduce to zero, and the cumulative regret would grow linearly.

5.2. The Exploration-Exploitation Trade-off

One of the central challenges in MAB problems is the trade-off between exploration and exploitation.

During the learning process, you always run into situations where you need to choose the next action based on prior observations. For example:



You have two major choices:

- **Exploration**: Trying different arms randomly, with the sole purpose to gather more information about their rewards.
- **Exploitation**: Selecting the arm that currently seems to offer the best reward based on the information gathered.

Next we discuss several basic strategies and analyze their pros and cons.

5.3. Greedy Strategy (Pure exploitation)

A greedy policy is to always choose the arm with the highest estimated payoff.
Specifically it selects the action based on

$$a_t = \arg\max_a \hat{R}_a(t) \tag{4}$$

Here, a_t is the action at time t, and $\hat{R}_a(t)$ is the estimated reward of arm a at time t:

$$\hat{R}_a(t) = \frac{\text{\# times alg pulled action and won up to time t}}{\text{\# times alg pulled action up to time t}}$$

While it's tempting to fully exploit the current knowledge (ie, estimated mean rewards), ignoring the uncertainty and blindly trusting the estimator can lead to getting stuck. Next we see that the greedy policy can fail when things don't go well.

Example: Two-Armed Bandit with Unlucky Initial Trials

Consider a simplified multi-armed bandit scenario with just two slot machines (arms), A and B. Each machine has a fixed and unknown probability of giving a reward:

- Bandit A: 40% chance of giving a reward.
- Bandit B: 60% chance of giving a reward.

The optimal solution is to always choose B. The greedy strategy chooses the machine with the highest observed average reward from previous plays, without additional exploration after the initial trials.

Let's say you start by playing each machine a limited number of times. Due to the inherent randomness in such a small sample size, the outcomes of these initial trials are:

- Bandit A: Gives a reward at least once (e.g., 50% observed success rate).
- Bandit B: Gives no reward (0% observed success rate).

When this happens, a greedy policy would consistently choose machine A for all subsequent plays because A had a history of generating rewards with > 0 probability and B had a record of 0 probability. Thus the greedy would get stuck at A and never try B again. However, this is a suboptimal choice.

This example illustrates how a greedy exploitation strategy can fail in a two-armed bandit scenario due to unlucky initial trials (which can happen with non-negligible probability). The lack of exploration in the strategy results in premature and incorrect conclusions about the true rewards. This example highlights the importance of incorporating **exploration** into decision-making strategies to avoid being misled by early random outcomes.

5.4. ε-Greedy Strategy

The ε-greedy strategy is to combine exploration and greedy strategy. It works as follows:

With probability ε, choose an arm at random (exploration), and with probability $1 - \varepsilon$, choose the arm with the highest estimated reward (exploitation).

If $\varepsilon = 1$, this strategy reduces to **pure exploration.**

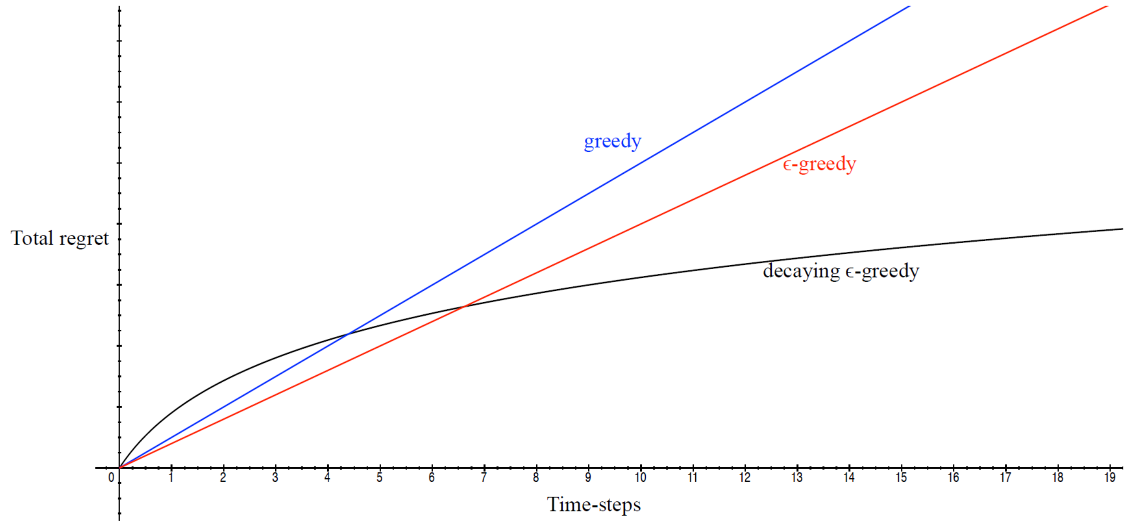
It is commonly used as a go-to heuristic in practice, for the following reasons:

- Simple to Implement: Easy to understand and implement.
- Balances Exploration and Exploitation: Ensures both aspects are addressed.
- Adjustable Exploration Rate: The parameter ε allows tuning of exploration.
- Generally Good Performance: Offers competitive performance in many (relatively simple) scenarios.

Despite these benefits, ε-greedy strategy are limited in many ways

- Inefficient Exploration: Random exploration can be suboptimal.
- Compromises Optimal Action: Always explores a fraction of the time, which can reduce overall rewards.
- Making ε decay over time can help, but it takes parameter tuning and hard to choose

Cumulative regret of greedy and epsilon-greedy



- If an algorithm forever explores it will have linear total regret
- If an algorithm never explores it will have linear total regret
- Is it possible to achieve sublinear total regret?

Recall the cumulative regret after T trials is:

$$\text{Regret}(T) = T \cdot \mu^* - \sum_{t=1}^{T} \mu_{a_t} \tag{5}$$

If $\textit{Regret}(T)$ grows linearly, ${Regret}(T) = O(T) $, it means that the learning algorithm never gets to learn to the action action.

We hope to make $\textit{Regret}(T)$ grow sublinearly, for example ${Regret}(T) = O(\sqrt{T}) $ would imply that the optimality gap decreases on the order of $O(1/\sqrt{T})$.

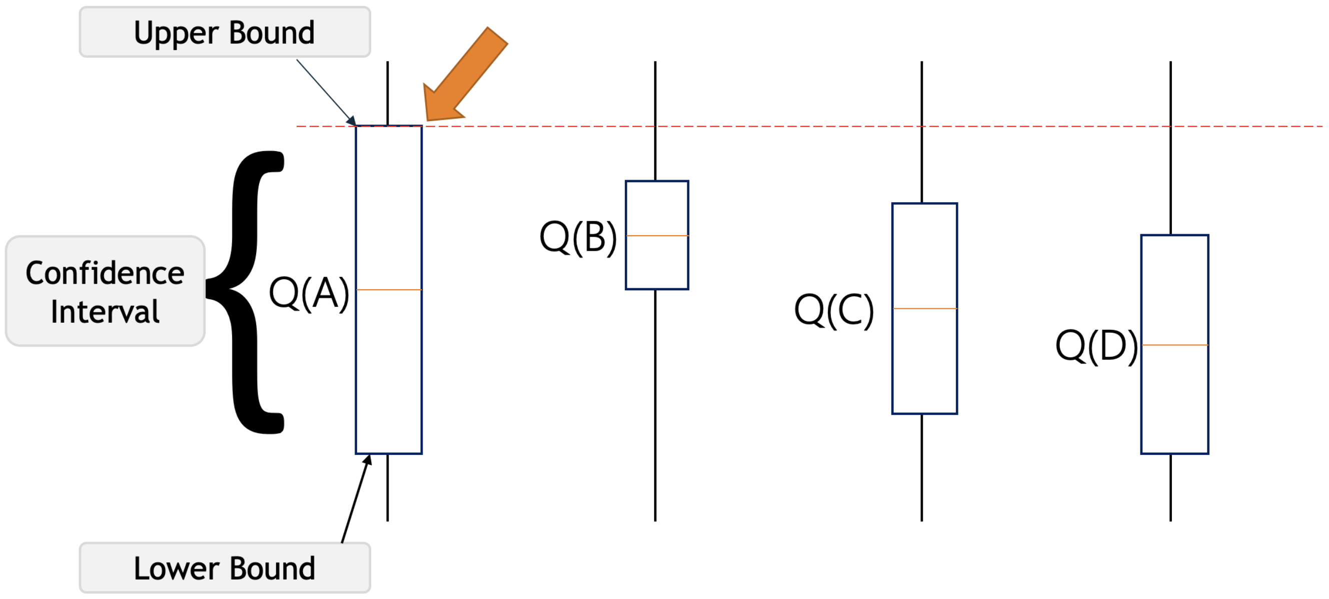
5.5. Balancing Exploitation-Exploration More Smartly: Upper Confidence Bound (UCB)

The UCB method mains an **optimistic estimate** for each arm's expected reward.

Suppose $\hat{R}_a(t)$ is the estimated reward for arm a at time t. The UCB selects arms based on a balance of estimated rewards and the uncertainty or variance in these estimates. The action selection is given by

$$A_t = \arg\max_a \left(\hat{R}_a(t) + \sqrt{\frac{2 \ln t}{N_a(t)}} \right) \tag{6}$$

Where $N_a(t)$ is the number of times arm a has been selected up to time t.



Optimism

Consider the simple case where the random payoff R_a for each arm a is a Bernoulli with parameter μ_a. Then $\hat{R}_a(t)$ is the mean of $N_a(t)$ i.i.d. Bernoulli random variables.

Let the upper confidence bound be

$$\overline{R}_a(t) := \hat{R}_a(t) + \sqrt{\frac{2 \ln t}{N_a(t)}}$$

We can try to derive the distribution of $\hat{R}_a(t)$. With probability $\geq 1 - 1/t$, we can show that

$$\left|\hat{R}_a(t) - \mu_a\right| \leq \sqrt{\frac{2 \ln t}{N_a(t)}}$$

and

$$\left|\overline{R}_a(t) - \mu_a\right| \leq 2\sqrt{\frac{2 \ln t}{N_a(t)}}$$

Therefore with high probability, probability, we get an **upper confidence bound** for the true mean reward:

$$\overline{R}_a(t) := \hat{R}_a(t) + \sqrt{\frac{2 \ln t}{N_a(t)}} \geq \mu_a$$

In other words, we always use **optimistic** estimates.

Regret of UCB (Optional)

Theorem
With close to 1 probability, the UCB algorithm has regret $Regret(T) \leq C\sqrt{AT \log(AT)}$, where C is some constant.

Proof sketch

Let's analyze the cumulative regret. With probability close to 1, we have

$$\begin{aligned}
\text{regret}(T) &= \sum_{t=1}^{T}(\mu_{a^*} - \mu_a(t)) \le \sum_{t=1}^{T}(\overline{R}_{a^*}(t) - \mu_a(t)) \quad \text{(by UCB)} \\
&\le \sum_{t=1}^{T}(\overline{R}_{a_t}(t) - \mu_a(t)) \quad \text{(algorithm is greedy w.r.t. } \tilde{r}^t \text{ at round \ensuremath\{t\})} \\
&\le (C \log AT) \sum_{t=1}^{T} \sqrt{\frac{1}{N_a(t)}}
\end{aligned}$$

Note that the term $(C \log AT)$ requires some probabilistic argument but it's a minor term, for which we omit the details.

Note that $N_a(t)$ is the number of times that arm a get pulled in the total t rounds, we have

$$\begin{aligned}
\sum_{t=1}^{T} \sqrt{\frac{1}{N_a(t)}} &= \sum_{a=1}^{A} \sum_{m=1}^{N_a(T)} \sqrt{\frac{1}{m}} \le \sum_{i=1}^{A} 2\sqrt{N_a(T)} \\
&\le 2\sqrt{\left(\sum_{i=1}^{A} 1\right)\left(\sum_{i=1}^{A} N_a(T)\right)} \qquad \text{(Cauchy-Schwarz)} \\
&= 2\sqrt{AT}
\end{aligned}$$

which completes the proof.

5.6. Aftermath of UCB

UCB models a simplified learning process, demonstrating how agents can learn from their environment in a trial-and-error manner, which is a cornerstone of RL. UCB provides key insights into basic concepts in Reinforcement Learning (RL). They are crucial for understanding more complex RL algorithms.

- first key insight is that the algorithm needs to estimate outcome of its decision. In UCB, the algorithm estimate the expectation of action's payoff by $\hat{R}_a(t)$, the empirical mean of payoffs from this action in the history. Later in RL, we see that whenever the RL agent chooses between actions, it also needs to predict/estimate the outcomes (namely value) of each action.
  Accurate value estimation is critical in RL for making informed decisions about which actions to take. For such a prediction problem, one may use supervised learning methods, instead of the empirical mean, to solve more complex RL problems.
- The second key insight is that UCB adds an exploration bonus to estimated reward, which measures the level of uncertainty in the current estimate. This bonus encourages the exploration of less-frequented arms. The exploration bonus in UCB encapsulates a fundamental challenge in RL - balancing exploration (trying new actions to discover their values) and exploitation (using known actions with high values). Later in RL, we will see that exploration is much harder in RL than in bandit problems. Effective RL algorithms need to find a balance between exploring new possibilities and exploiting known rewards.