Bandit Problems and Online Learning

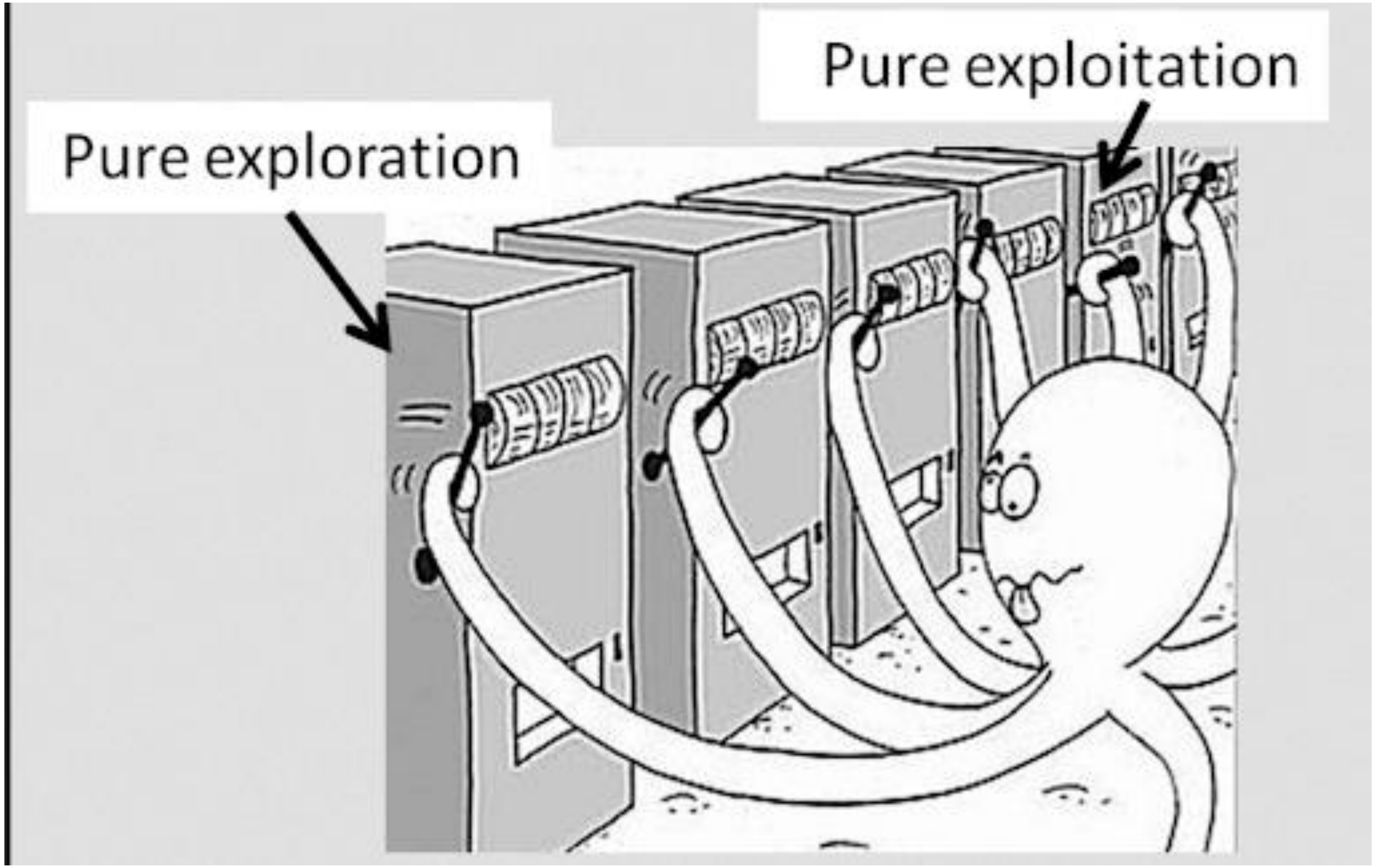
Reinforcement Learning : Module 02

The slides contain

- Excerpts from the book Reinforcement Learning : An Introduction, 2nd edition, Richard S. Sutton and Andrew G. Barto
- Content acquired using ChatGPT

Armed Bandit Problem

Bandit Problem



Armed Bandit : Introduction

- Selecting a college
  - Explore
  - Exploit
- Working a new project
  - Explore
  - Exploit

Selecting a college:
Explore: This could represent researching various colleges, considering different factors like location, programs, finances, etc.
Exploit: This could represent choosing a college based on past information, such as recommendations from friends or family, reputation, or personal preference.

Working on a new project:
Explore: This could represent trying different approaches, experimenting with various techniques, and gathering information early on.
Exploit: This could represent focusing on proven methods, utilizing successful strategies from past projects, or refining existing ideas.

Key Points:
Both scenarios involve making choices with unknown outcomes.
There's a trade-off between exploration (trying new things, potentially discovering better options) and exploitation (focusing on what's known to work, achieving results quickly).
The "best" approach might involve a balance of both strategies.

Further thoughts:
The slide seems to suggest that applying the "Bandit Problem" framework can be helpful in making informed decisions even in non-technical situations.
It raises interesting questions about how much exploration is necessary and how to determine the right balance between exploring and exploiting in different contexts.

The slide introduces the concept of the multi-armed bandit problem, a classic scenario in reinforcement learning.

https://youtu.be/e3L4VocZnnQ?si=Dz9U_lXXGFi8Gux5

Armed Bandit : Introduction

- 3 Restaurant Example
  - Average Happiness 10, 8 and 5
  - 300 Days to visit

3 Restaurant Example: A simplified scenario with three restaurants – each has an average happiness rating (10, 8, and 5). This represents a set of options with varying, but unknown, rewards.
300 Days to Visit: This represents the number of decisions the agent (you in this example) would get to make.

- Explore Only
  - Average Happiness : 2300, Regret : 700

If you only explore, you'd try all three restaurants over time, gaining an understanding of which is best.
Average Happiness: 2300 (a rough estimate assuming a perfect split of time between restaurants)
Regret: 700 (the difference between the optimal outcome and what you likely achieved with pure exploration).

- Exploit Only
  - Day 1 : R1 → 5, Day 2 : R2 → 8, Day 3 : R3 → 5
  - So we continue with R2 for rest 297 Days

Day 1: Choose R1, get a happiness level of 10
Day 2: Choose R2, get a happiness level of 8.
Day 3: Choose R3, get a happiness level of 5.
Based on those initial days, you'd likely exploit R2 for the rest of the time because it offers the second-highest observed reward.

- ε Greedy : 10%

A common strategy for the bandit problem.
10% suggests that 10% of the time, you'll take a random action (exploration), and 90% of the time, you'll choose the option with the currently highest known reward (exploitation).

Exploitation: Focus on your current knowledge of rewards to maximize immediate gains.
Exploration: Try out different options to potentially discover even better rewards in the long run.
The optimal strategy often involves striking a balance between exploring and exploiting.
The epsilon-greedy strategy is a simple but effective way to approach this balance.
The bandit problem is a model, useful for thinking about broader decision-making scenarios where you have limited tries to maximize your outcome.

Definition: It's a foundational problem in reinforcement learning, involving making optimal choices when faced with limited information.
Analogy: It uses the metaphor of a slot machine with multiple levers (k arms) representing different options.
Goal: The agent aims to maximize its cumulative reward by choosing the best action (lever) over time.

k-armed Bandit

Significance:
It highlights the exploration-exploitation dilemma: whether to try new options (explore) or stick with what seems good (exploit).
It serves as a building block for more complex reinforcement learning problems.

- A *k*-armed bandit is a simple but fundamental problem in the field of reinforcement learning and decision theory.
- The term is derived from the idea of a slot machine or "one-armed bandit," with the extension of having *k* arms, each corresponding to a different action.
- The problem is used to study how an agent can learn to maximize its cumulative reward over time by selecting actions.
- The problem serves as a foundational concept in reinforcement learning, providing insights into the exploration-exploitation dilemma.
- Solutions to the *k*-armed bandit problem form the basis for more complex problems in reinforcement learning, where agents face larger state and action spaces.

k-armed Bandit : Key Elements

Slot Machine (k-armed bandit): The decision-making scenario is modeled as a slot machine with multiple levers (k arms) representing different possible actions.
Actions: Each arm corresponds to a distinct action the agent can take.

- **Setup:**
  - There is a slot machine (bandit) with *k* arms.
  - Each arm represents a different action that the agent can take.

Fixed distributions allow for simpler strategies, as rewards remain predictable.
Changing distributions require more sophisticated approaches to adapt to the changing environment and find the optimal arm dynamically.
Marketing Campaigns: Different channels (arms) like email, social media, or ads could have varying success rates (rewards) depending on the target

- **Reward Distribution:**
  - Pulling an arm results in a numerical reward.
  - Each arm has an associated probability distribution of rewards. These distributions can be fixed or changing over time.
- **Agent's Objective:**
  - The agent's goal is to learn which arm(s) yield the highest average reward over time.
  - The agent must balance exploration (trying different arms to discover their rewards) and exploitation (choosing the arm with the highest expected reward based on current knowledge).

k-armed Bandit : Key Elements

- **Exploration-Exploitation Tradeoff:**
  - In the early stages, the agent needs to explore by trying different arms to estimate their reward distributions.
  - As the agent gathers more information, it transitions toward exploitation, focusing on the arms with higher expected rewards.
- **Learning Mechanism:**
  - The agent employs a learning mechanism to update its estimates of the reward distributions for each arm based on the outcomes of previous actions.
  - Common methods for updating estimates include sample averages or more sophisticated algorithms such as the epsilon-greedy strategy.
- **Regret:**
  - The regret in the context of a *k*-armed bandit is the difference between the expected cumulative reward obtained by the agent and the maximum possible cumulative reward achievable by always choosing the best arm.

As the owner of an online streaming service, your goal is to recommend videos that maximize user engagement, which could translate to metrics like watch time, repeat views, or positive user feedback. The k-armed bandit problem is a perfect framework for approaching this challenge. Here's a detailed strategy you could follow:

1. Define Arms and Rewards:

Arms: Each video in your library can be considered an arm. Rewards: User engagement metrics like watch time or positive ratings can be used as rewards. You could also consider a combination of these metrics.

k-armed Bandit : Case Study

2. Exploration-Exploitation Strategy:

Initially: Employ a high exploration rate. This means recommending a diverse range of videos to different users, regardless of their previous viewing history. This helps gather data on how users engage with different types of content.

- Imagine you are the owner of an online streaming service with a vast library of videos (content), and you want to recommend videos to your users to maximize their engagement.
- Describe in detail which strategy would you follow and why?

3. Learning Mechanism:

Sample Average: You can use the sample average of each video's engagement metric (watch time, ratings) based on past user interactions as a basic learning mechanism.

4. Recommendation Strategy:

Hybrid Approach: Combine exploitation and exploration in your recommendations:

Exploit: A portion of recommendations should be based on the highest-performing videos (arms) identified from user data. This ensures users get access to content they're likely to enjoy based on their past preferences.

Explore: Dedicate a smaller portion of recommendations to explore less-viewed videos (arms) with high potential, based on factors like genre, actors, or popularity on other platforms. This helps discover hidden gems and cater to divers user interests.

5. Dynamic Adjustments:

Continuously monitor user engagement data.

Adapt your exploration rate: As you gather more information, you can gradually decrease exploration and increase exploitation, focusing on recommendations with proven success.

Consider user context: Personalize recommendations based on user demographics, viewing history, and current preferences.

Why this strategy?

This approach balances the need to cater to existing user preferences while also discovering new content that might resonate with a wider audience. This helps in:

Maximizing overall user engagement.

Maintaining user interest by offering diverse content.

Identifying potential breakout videos that might become future user favorites.

It's important to note that this is a general framework, and the specific details of your strategy might need to be adjusted based on factors like the size and diversity of your video library, the specific engagement metrics you prioritize, an preferences of your user base.

Armed Bandit (Problem) Types

1-Armed Bandit

- In a 1-armed bandit problem, there is a single action or choice available to an agent. The term is derived from the idea of a slot machine or "one-armed bandit" with a single lever or arm.
- The agent repeatedly chooses the same action and receives a numerical reward each time.
- The challenge is to learn the value of that single action to maximize cumulative rewards.

1-Armed Bandit : Examples

- In online advertising, A/B testing involves comparing two versions (A and B) of an ad to determine which one performs better in terms of click-through rates or conversions. Each version of the ad is akin to a "lever" or action in the 1-armed bandit scenario. The goal is to identify the version that yields the highest engagement (reward).
- In web development and e-commerce, A/B testing involves comparing two versions of a webpage to determine which one leads to higher conversion rates (e.g., more sign-ups, purchases). Each webpage version is analogous to a "lever" or action in the 1-armed bandit scenario. The goal is to identify the webpage version that maximizes user interactions or conversions.

1-Armed Bandit : Examples

- In email marketing, marketers often test different subject lines to determine which one leads to higher open rates.
- Developers might test different icons for a mobile app to see which one attracts more downloads and user engagement.
- Advertisers may test different variations of a web banner ad to determine which one performs better in terms of click-through rates.
- Website designers may experiment with different colors, texts, or placements for call-to-action buttons to optimize user interactions.
- E-commerce platforms might test different pricing strategies for a product to determine the optimal price point for maximizing sales.

K-Armed Bandit

- In a k-armed bandit problem, there are
  - k different actions or choices available to an agent. Each action corresponds to a different arm of the bandit.
  - The agent faces the challenge of learning the values of all k actions and deciding which action to choose at each time step to maximize cumulative rewards.

1. Multiple Actions:
The agent is presented with k distinct actions (choices). These actions could be anything from choosing a lever on a slot machine to selecting a marketing strategy or picking an investment option.
Each action corresponds to a "arm" of the metaphorical "bandit."

2. Learning and Maximizing Rewards:
The challenge for the agent lies in learning the values of all k actions. This means understanding the true potential reward associated with each action.
Since the rewards might not be readily available upfront, the agent needs to learn through interaction.

The ultimate goal is to maximize the cumulative reward over time by choosing the action with the highest expected value in each decision step.
This basic framework lays the foundation for the core dilemma at the heart of the k-armed bandit problem: balancing exploration and exploitation. The agent needs to explore different actions (arms) to learn their true values, while also exploiting its current knowledge to choose the best option known so far.

K-Armed Bandit : Examples

- Medical Treatment Selection
  - In a medical context, a patient may have k different treatment options for a specific condition, each with varying effectiveness and potential side effects. Each treatment option is a potential "arm" in the k-armed bandit problem. The goal is to identify the treatment that yields the best patient outcomes.
- Algorithm Selection for Image Recognition
  - In machine learning, researchers may experiment with k different algorithms for image recognition to find the one that performs best on a given dataset. Each algorithm is a potential "arm" in the k-armed bandit problem. The goal is to identify the algorithm that achieves the highest accuracy on the task.

K-Armed Bandit : Examples

- Routing in Computer Networks
  - In a computer network, data packets can be routed through k different paths to reach a destination. Each path may have different latency and reliability characteristics. Each routing path represents a potential "arm" in the k-armed bandit problem. The goal is to choose the path that minimizes latency and ensures reliable data delivery.
- Game Strategy Selection
  - In a strategy game, a player may have k different strategies to choose from, each with its strengths and weaknesses. Each strategy is a potential "arm" in the k-armed bandit problem. The player aims to identify the strategy that maximizes their chances of winning.

Multi-Armed Bandit

- The term "multi-armed bandit" is a more general concept that includes scenarios with more than one bandit, each having its set of arms or actions.
- In a multi-armed bandit scenario, the agent must decide which bandit to interact with and which action to choose within the selected bandit. Different bandits may have different reward distributions for their arms.
- The focus is on decision-making under uncertainty when faced with multiple options, and the agent needs to explore and exploit to learn the optimal actions in each bandit.

Multi-Armed Bandit : Examples

- News Article Recommendation:
  - In an online news platform, a user is presented with multiple articles from different categories (e.g., politics, sports, entertainment). The platform aims to recommend articles that maximize user engagement. Each article category represents an "arm" in the multi-armed bandit problem. The goal is to dynamically select article categories that lead to higher user click-through rates and reading time.
- Dynamic Pricing in E-Commerce:
  - An e-commerce website sells a product with different pricing strategies. The website aims to dynamically adjust prices to maximize revenue while considering customer satisfaction. Each pricing strategy is like an "arm" in the multi-armed bandit problem. The goal is to find the pricing strategy that leads to the highest conversion rates and revenue.

Multi-Armed Bandit : Examples

- Online Ad Placement:
  - An online advertising platform has multiple ad placements on a webpage. The platform aims to maximize user clicks and conversions by dynamically selecting the best ad placement for each user. Each ad placement is a potential "arm" in the multi-armed bandit problem. The goal is to identify the ad placement that yields the highest user engagement.
- Resource Allocation in Cloud Computing:
  - In a cloud computing environment, multiple servers or resources are available to handle incoming requests. The system aims to dynamically allocate resources to minimize response times and maximize resource utilization. Each resource allocation strategy is like an "arm" in the multi-armed bandit problem. The goal is to choose the allocation strategy that optimizes system performance.

Action Value Methods

Action Value Methods

1. Action Value:
It represents the expected future reward an agent can obtain by taking a specific action in a particular state.
It essentially tells the agent how "good" it is to take a specific action in a specific situation.

- The action value is a measure of the expected cumulative reward an agent can achieve by taking a specific action in a particular state.
- Action-value methods involve estimating the values of different actions

2. Estimating Action Values:
Action-value methods focus on estimating the value of different actions available to the agent in each state.

  - in order to make use of these estimates for action selection decisions.

3. Using Action Values for Decisions:
Once the agent has an estimate of the action values for different actions in a given state, it can use these estimates to make informed decisions about which action to take.
This typically involves choosing the action with the highest estimated value, aiming to maximize long-term rewards.

- From the Book
  - *We begin by looking more closely at methods for estimating the values of actions and for using the estimates to make action selection decisions, which we collectively call action-value methods.*

Common Action-Value Methods: Some common action-value methods include Q-learning, SARSA, and Deep Q-Networks (DQNs).
Exploration vs. Exploitation: While choosing the action with the highest estimated value often leads to higher future rewards, it's important to balance this with exploration. This means the agent should also try different actions, even if their estimated values are lower, to potentially discover better options in the long run.

Action Value Methods : Sample-Average Method

- Recall that the true value of an action is the mean reward when that action is selected.
- One natural way to estimate this is by averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

where $\mathbb{1}_{predicate}$ denotes the random variable that is 1 if *predicate* is true and 0 if it is not. If the denominator is zero, then we instead define $Q_t(a)$ as some default value, such as 0. As the denominator goes to infinity, by the law of large numbers, $Q_t(a)$ converges to $q_*(a)$. We call this the *sample-average* method for estimating action values because each estimate is an average of the sample of relevant rewards.

Action Value Methods

- Sample-Average method this is just one way to estimate action values, and not necessarily the best one.
- Two common action-value methods are
  - Monte Carlo methods
  - Temporal Difference (TD) methods.
- Nevertheless, for now let us stay with this simple estimation method
  - **and turn to the question of how the estimates might be used to select actions.**

Selecting Action

- The simplest action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions.
- If there is more than one greedy action, then a selection is made among them in some arbitrary way, perhaps randomly.
- We write this greedy action selection method as

$$A_t \doteq \underset{a}{\arg\max}\, Q_t(a)$$

Selecting Action

- Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better.
- A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ε, instead select randomly from among all the actions with equal probability, independently of the action-value estimates.
- We call methods using this near-greedy action selection rule ε-greedy methods.

ε-Greedy

- The *ε*-greedy strategy is a common exploration-exploitation strategy used in reinforcement learning, particularly in the context of *k*-armed bandit problems and Q-learning.
- The strategy balances the agent's need to explore (try different actions to learn about their rewards) and exploit (choose actions with the highest expected rewards based on current knowledge).

ε-Greedy

- **Exploration Phase (*ε* Exploration):**
  - With probability *ε* (epsilon), the agent explores by selecting a random action from the available set of actions.
  - This random exploration allows the agent to gather information about the rewards associated with different actions.
- **Exploitation Phase ((1-*ε*) Exploitation):**
  - With probability $1-\varepsilon$, the agent exploits by selecting the action with the highest estimated value based on its current knowledge.
  - The estimated values are typically derived from a Q-table (for Q-learning) or an action-value function.

ε-Greedy

- The *ε*-greedy strategy introduces a trade-off between exploration and exploitation.
- A higher *ε* encourages more exploration, while a lower *ε* emphasizes exploitation.
- The choice of *ε* depends on the specific problem and learning scenario.
- A commonly used value for *ε* is 0.1, meaning there is a 10% chance of exploration and a 90% chance of exploitation.

ε-Greedy : 10-Armed Bandit Performance

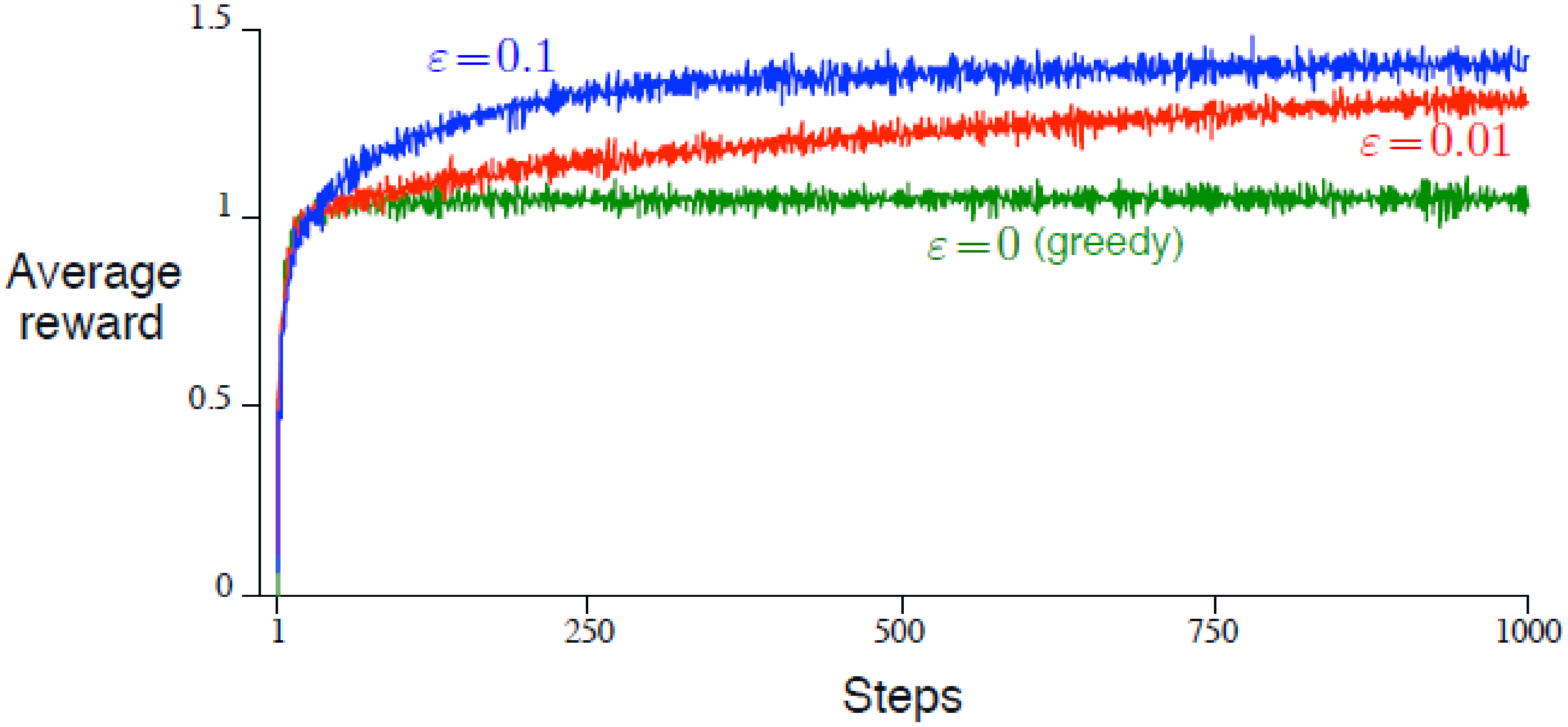


Figure : Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

ε-Greedy : 10-Armed Bandit Performance

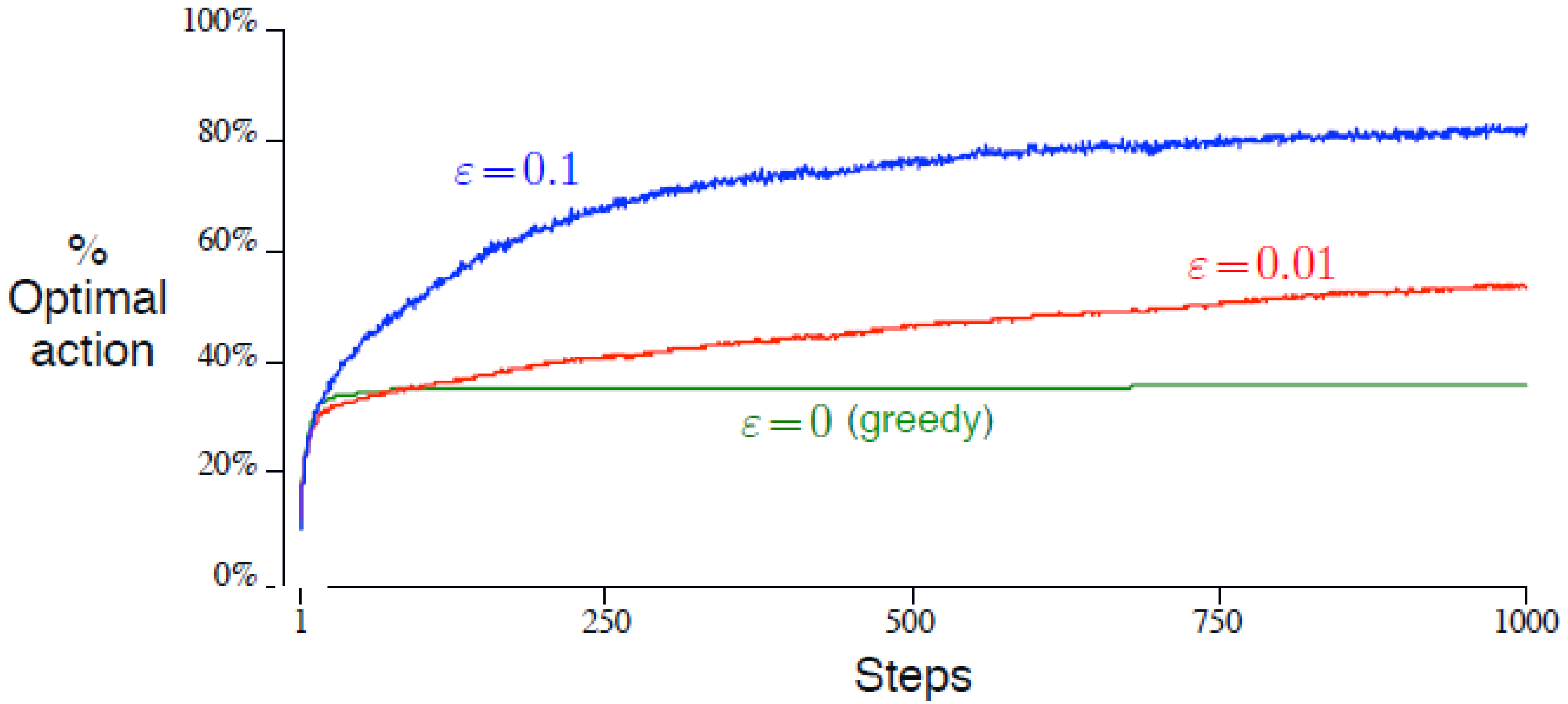


Figure : Average performance of ε-greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

One More Time : Armed Bandit Example

- 3 Restaurant Example
  - Average Happiness 10, 8 and 5
  - 300 Days to visit
- Explore Only
  - Average Happiness : 2300, Regret : 700
- Exploit Only
  - Day 1 : R1 → 5, Day 2 : R2 → 8, Day 3 : R3 → 5
  - So we continue with R2 for rest 297 Days
- ε Greedy : 10%

Bandit Example

- Bandit example Consider a k-armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using ε-greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a. Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = -1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = -2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$.
- On some of these time steps the ε case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred?

(Mentioned on Page 30 of the book)

the incremental implementation approach offers a straightforward and efficient way to update action values in multi-armed

Incremental Implementation

- Recall $$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$
- Let us try finding how these averages can be computed in a computationally efficient manner, in particular, with constant memory and constant per-time-step computation.
- To simplify notation we concentrate on a single action. Let R_i now denote the reward received after the i^{th} selection of this action, and let Q_n denote the estimate of its action value after it has been selected $n - 1$ times, which we can now write simply as

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

Incremental Implementation

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

- The obvious implementation would be to maintain a record of all the rewards and then perform this computation whenever the estimated value was needed.
- However, if this is done, then the memory and computational requirements would grow over time as more rewards are seen.
- Each additional reward would require additional memory to store it and additional computation to compute the sum in the numerator.
- It is easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward.

Incremental Implementation

- Given Q_n and the n^{th} reward, R_n, the new average of all n rewards can be computed by

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

- It also holds for n = 1, obtaining $Q_2 = R_1$ for arbitrary Q_1
- This implementation requires memory only for Q_n and n, and only the small computation for each new reward.

Qn: Represents the estimated action value after the action has been chosen n - 1 times.
Rn: Represents the reward received after the n-th time the action is selected.
New Qn+1: Represents the updated estimated action value after considering the n-th reward.

- The general form is

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \left[T \right.$$

- The expression ***Target−OldEstimate*** is an error in the e
- It is reduced by taking a step toward the "Target." The target is presumed to indicate a desirable direction in which to move, though it may be noisy. In the case above, for example, the target is the n^{th} reward.
- The step-size parameter (***StepSize***) used in the incremental method changes from time step to time step. In processing the n^{th} reward for action a, the method uses the step-size parameter 1/n.
- The step-size parameter is denoted by α or, more generally, by $\alpha_t(a)$.

StepSize (α) for Q-Learning and Incremental Method

- The key differen[redacted]cremental methods lies in the specific update rules used. Q-learning involves updating the action-value estimates based on the immediate reward and the maximum estimated value of the next state, while incremental methods generally update state value estimates based on the observed return.
- In practice, the choice of the learning rate (α) in both Q-learning and incremental methods is crucial, as it affects the balance between exploration and exploitation, the stability of learning, and convergence properties. The optimal choice of α may vary depending on the specific problem and characteristics of the environment.

A simple bandit algorithm

- Pseudocode for a complete bandit algorithm using incrementally computed sample averages and ε-greedy action selection is shown below.
  - The function bandit(a) is assumed to take an action and return a corresponding reward.

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \text{argmax}_a\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Tracking a Nonstationary Problem

A bandit problem, also known as a multi-armed bandit problem, is a fundamental problem in reinforcement learning and decision theory. It describes a situation where you have a set of choices (called arms or actions) that you can exploit, but you don't know which choice is the best upfront. You learn by interacting with the choices and receiving rewards. The challenge lies in balancing exploration (trying new options to find potentially better ones) and exploitation (choosing the option that currently seems best based on past rewards).

Tracking a Nonstationary Problem

Multi-armed bandit: The most basic type, where you don't have any prior knowledge about the rewards of each option.
Non-stationary bandit: The rewards for each option can change over time.

Non-stationary RL problems deal with a more dynamic environment where the optimal strategy needs to be continuously adjusted based on significant changes in the underlying dynamics.

- The averaging methods discussed so far are appropriate for stationary bandit problems, that is, for bandit problems in which the reward probabilities do not change over time.
- We often encounter reinforcement learning problems that are effectively nonstationary.
- In such cases it makes sense to give more weight to recent rewards than to long-past rewards.
- Recall

$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$

  - As n increases the weight of $[R_n - Q_n]$ decreases

Tracking a Nonstationary Problem

- To give more weight to recent rewards, popularly, a constant step-size parameter is used.
- Thereby

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$

- where the step-size parameter α ∈ (0, 1] is constant
  - i.e. $0 < \alpha \leq 1$

Tracking a Nonstationary Problem

- This results in Q_{n+1} being a weighted average of past rewards and the initial estimate Q_1

$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.
\end{aligned}$$

What’s Weighted Average?

- A weighted average is a type of average where different values are given different levels of importance or weight in the overall calculation.
- In a standard (unweighted) average, all values contribute equally to the result. However, in a weighted average, some values have more influence than others based on their assigned weights.
- The formula for calculating a weighted average is:

$$\text{Weighted Average} = \text{Sum of } (\text{Value} \times \text{Weight}) \ / \ \text{Sum of Weights}$$

- Here:
  - "Value" refers to each individual value in the set.
  - "Weight" refers to the corresponding weight assigned to each value.
  - The sum is taken over all values in the set.

This weight decays exponentially with the number of steps (n-i) that have passed since the reward was received. In simpler terms, the influence of older rewards diminishes as the agent gathers more recent experiences.

Tracking a Nonstationary Problem

- We call this a weighted average because the sum of the weights is

$$(1-\alpha)^n + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} = 1$$

- Note that the weight, $\alpha(1-\alpha)^{n-i}$, given to the reward R_i depends on how many rewards ago, $n-i$, it was observed.
- The quantity $1-\alpha$ is less than 1, and thus the weight given to R_i decreases as the number of intervening rewards increases.
- In fact, the weight decays exponentially according to the exponent on $1-\alpha$.
  - If $1-\alpha = 0$, then all the weight goes on the very last reward, Rn, because of the convention that $0^0 = 1$.
- Accordingly, this is sometimes called an *exponential recency-weighted average*.

Tracking a Nonstationary Problem

- Sometimes it is convenient to vary the step-size parameter from step to step.
- Let $\alpha_n(a)$ denote the step-size parameter used to process the reward received after the n^{th} selection of action a.
- The choice $\alpha_n(a) = 1/n$ results in the sample-average method, which is guaranteed to converge to the true action values by the law of large numbers.
- But of course convergence is not guaranteed for all choices of the sequence $\{\alpha_n(a)\}$.

Tracking a Nonstationary Problem

- A well-known result in stochastic approximation theory gives us the conditions required to assure convergence with probability 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

- The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations.
- The second condition guarantees that eventually the steps become small enough to assure convergence.

Tracking a Nonstationary Problem

- Note that both convergence conditions are met for the sample-average case $\alpha_n(a) = 1/n$.
- But not for the case of constant step-size parameter, $\alpha_n(a) = \alpha$, the second condition is not met, indicating that the estimates never completely converge but continue to vary in response to the most recently received rewards.
- This is actually desirable in a nonstationary environment, and problems that are effectively nonstationary are the most common in reinforcement learning.

Tracking a Nonstationary Problem

- Also sequences of step-size parameters that meet both the conditions often converge very slowly or need considerable tuning in order to obtain a satisfactory convergence rate.
- Although sequences of step-size parameters that meet these convergence conditions are often used in theoretical work, they are seldom used in applications and empirical research.

Programming Exercise

Exercise 2.5 (programming) Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the 10-armed testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say by adding a normally distributed increment with mean zero and standard deviation 0.01 to all the $q_*(a)$ on each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\varepsilon = 0.1$ and longer runs, say of 10,000 steps. □

(Mentioned on Page 33 of the book)

https://youtu.be/apwob9KzG90?si=REI9TsJ7fcc8T-8s

Optimistic Initial Values

- The methods we have discussed so far are dependent to some extent on the initial action-value estimates, $Q_1(a)$.
- In the language of statistics, these methods are biased by their initial estimates.
- For the sample-average methods, the bias disappears once all actions have been selected at least once, but for methods with constant α, the bias is permanent, though decreasing over time.
- In practice, this kind of bias is usually not a problem and can sometimes be very helpful. While initial Q-value estimates can introduce bias, they can also be a valuable tool
- The downside is that the initial estimates become, in effect, a set of parameters that must be picked by the user (But generally all are set to zero).
- The upside is that they provide an easy way to supply some prior knowledge about what level of rewards can be expected.

Optimistic Initial Values

- Initial action values can also be used as a simple way to encourage exploration.
- Suppose that instead of setting the initial action values to zero, we set them all to a number that is wildly optimistic +5
- This optimism encourages action-value methods to explore. Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being “disappointed” with the rewards it is receiving.
- The result is that all actions are tried several times before the value estimates converge.
- The system does a fair amount of exploration even if greedy actions are selected all the time.

Optimistic Initial Values

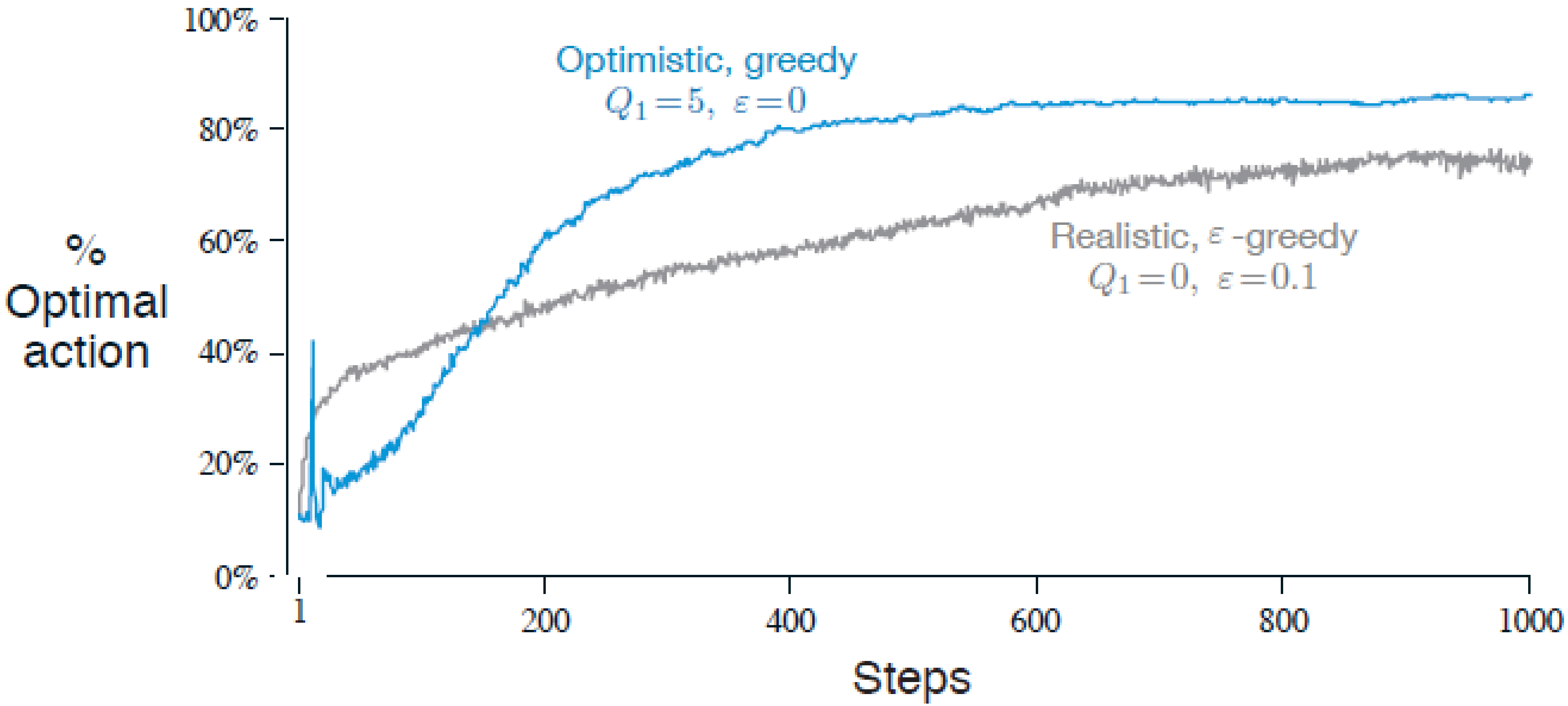


Figure : The effect of optimistic initial action-value estimates on the 10-armed testbed. Both methods used a constant step-size parameter, $\alpha = 0.1$.

Optimistic Initial Values

- Figure shows the performance on the 10-armed bandit testbed of a greedy method using $Q_1(a) = +5$, for all a.
- For comparison, also shown is an ε-greedy method with $Q_1(a) = 0$.
- Initially, the optimistic method performs worse because it explores more, but eventually it performs better because its exploration decreases with time.
- We call this technique for encouraging exploration optimistic initial values.
- We regard it as a simple trick that can be quite effective on stationary problems, but it is far from being a generally useful approach to encouraging exploration.
- For example, it is not well suited to nonstationary problems
    - because its drive for exploration is inherently temporary.

Upper-Confidence-Bound Action Selection

Upper-Confidence-Bound : Need

- Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates.
- The greedy actions are those that look best at present, but some of the other actions may actually be better.
- ε-greedy action selection forces the non-greedy actions to be tried, but indiscriminately, with no preference for those that are nearly greedy or particularly uncertain.
- It would be better to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.

Upper-Confidence-Bound : Formula

- One effective way of doing this is to select actions according to

$$A_t \doteq \underset{a}{\arg\max}\left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

- Where
  - $\ln t$ denotes the natural logarithm of t (the number that e ≈ 2.71828 would have to be raised to in order to equal t),
  - $N_t(a)$ denotes the number of times that action a has been selected prior to time t
  - the number c > 0 controls the degree of exploration.
  - If $N_t(a) = 0$, then a is considered to be a maximizing action.

Upper-Confidence-Bound Action Selection

- The square-root term is a measure of the uncertainty or variance in the estimate of a's value.
- The quantity being max'ed over is thus a sort of upper bound on the possible true value of action a, with c determining the confidence level.
- Each time a is selected the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.
- On the other hand, each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.
- The use of the natural logarithm means that the increases get smaller over time, but are unbounded.
- All actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.

Upper-Confidence-Bound : Performance

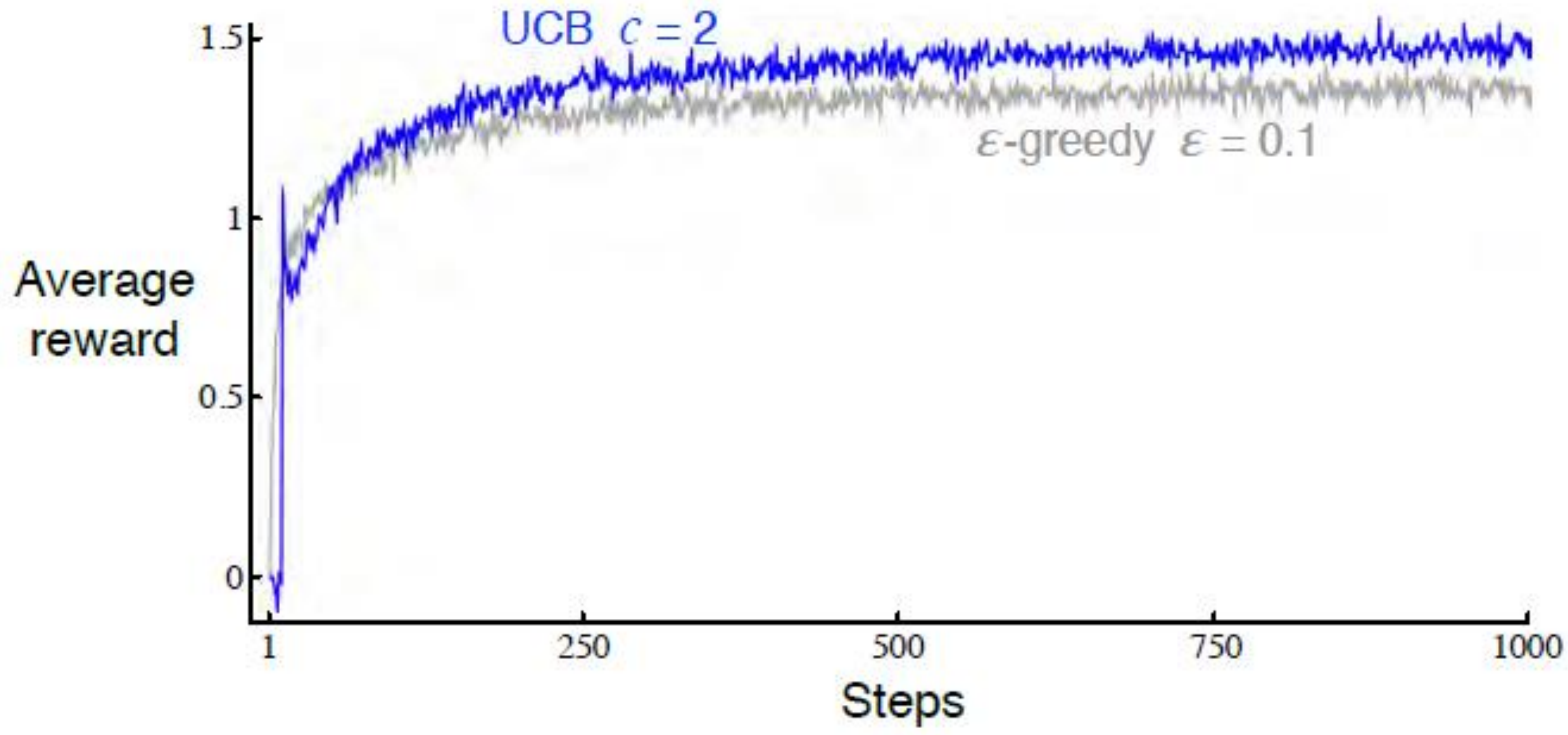


Figure : Average performance of UCB action selection on the 10-armed testbed. As shown, UCB generally performs better than ε-greedy action selection, except in the first k steps, when it selects randomly among the as-yet-untried actions

Upper-Confidence-Bound : Comments

- UCB often performs well, as shown in the figure, but is more difficult than ε-greedy to extend beyond bandits to the more general reinforcement learning settings.
- Another difficulty is dealing with large state spaces, particularly when using function approximation.
- In these more advanced settings the idea of UCB action selection is usually not practical.

Gradient Bandit Algorithms

Gradient : The gradient provides information about the rate of change or slope of a function at a particular point. For a function of two variables, the gradient points in the direction of the steepest ascent on the surface defined by that function.

Gradient Bandit Algorithms

- So far we have considered methods that estimate action values and use those estimates to select actions. This is often a good approach, but it is not the only one possible.
- Let us consider learning a numerical preference for each action a, which we denote $H_t(a)$.
- The larger the preference, the more often that action is taken, but the preference has no interpretation in terms of reward.

Gradient Bandit Algorithms

- Only the relative preference of one action over another is important; if we add 1000 to all the action preferences there is no effect on the action probabilities, which are determined according to a soft-max distribution as follows:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- *Where $\pi_t(a)$*, is the probability of taking action a at time t.
- Initially all action preferences are the same (e.g., *$H_1(a) = 0$*, for all a) so that all actions have an equal probability of being selected

Softmax is a mathematical function that takes a vector of arbitrary real-valued scores and transforms them into a probability distribution.

Gradient Bandit Algorithms

- There is a natural learning algorithm for this setting based on the idea of stochastic gradient ascent.
- On each step, after selecting action A_t and receiving the reward R_t, the action preferences are updated by:

$$
\begin{aligned}
H_{t+1}(A_t) &\doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), && \text{and} \\
H_{t+1}(a) &\doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), && \text{for all } a \neq A_t
\end{aligned}
$$

Updated preference for action a (other than the chosen action) at the next time step (t+1).

- where $\alpha > 0$ is a step-size parameter, and $\bar{R}_t \in \boldsymbol{R}$ is the average of all the rewards up through and including time t, which can be computed incrementally.
  - P. N. $\bar{R}_t$ is calculated differently for stationery and non-stationery problems.

Gradient Bandit Algorithms

- The $\overline{R}_t$ term serves as a baseline with which the reward is compared.
- If the reward is higher than the baseline, then the probability of taking A_t in the future is increased, and if the reward is below baseline, then probability is decreased.
- The non-selected actions move in the opposite direction.

Gradient Bandit Algorithms Performance

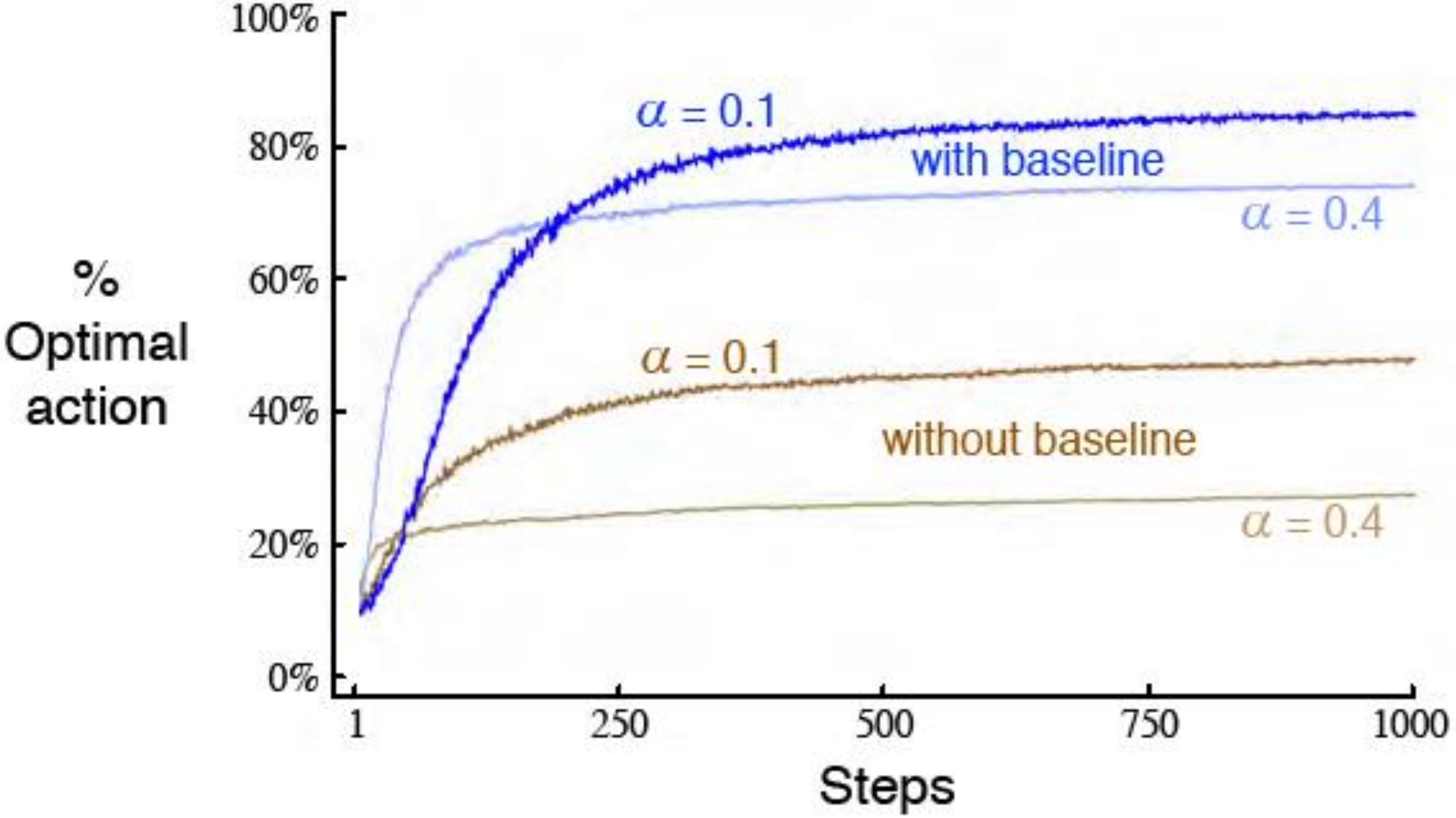


Figure : Average performance of the gradient bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near +4 rather than near zero

Gradient Bandit Algorithms Performance

- Figure shows results with the gradient bandit algorithm on a variant of the 10-armed testbed in which the true expected rewards were selected according to a normal distribution with a mean of +4 instead of zero (and with unit variance as before).
- This shifting up of all the rewards has absolutely no effect on the gradient bandit algorithm because of the reward baseline term, which instantaneously adapts to the new level.
- But if the baseline were omitted, then performance would be significantly degraded, as shown in the figure.

End