Deep Reinforcement Learning

2025-56 First Semester, M.Tech (AIML)

Session #2-3:
Multi-armed Bandits

Instructors - DRL Course

Agenda for the class

- Recap
- k-armed Bandit Problem & its significance
- Action-Value Methods
  - Sample Average Method & Incremental Implementation
- Non-stationary Problem
- Initial Values & Action Selection

Acknowledgements: Some of the slides were adopted *with permission* from the course CSCE-689 (Texas A&M University) by Prof. Guni Sharon

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

  Reward is chosen from a stationary probability distribution that depends on the selected action
- **Objective** : to *maximize the expected total reward over some time period*

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

  Reward is chosen from a stationary probability distribution that depends on the selected action
- **Objective** : to *maximize the expected total reward over some time period*

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

  Reward is chosen from a stationary probability distribution that depends on the selected action
- **Objective** : to *maximize the expected total reward over some time period*

Strategy:

- Identify the best lever(s)
- Keep pulling the identified ones

Questions:

- How do we define the *best ones*?
- What are the best levers?

K-armed Bandit Problem

Problem

- You are faced repeatedly with a choice among k different options, or actions
- After each choice of actions you receive a numerical reward

  Reward is chosen from a stationary probability distribution that depends on the selected action
- **Objective** : to *maximize the expected total reward over some time period*

$\mathbb{E}[a] = -\$0.5$ $\mathbb{E}[b] = -\$0.2$ $\mathbb{E}[c] = \$0.1$ $\mathbb{E}[d] = \$0.11$

If pressing lever 'a' in a bandit machine gives rewards of {1, 2, 3} with an average of 2, then

$$q_*(A) = 2$$

- ***Expected Mean Reward*** for each action selected
  → call it ***Value*** of the action

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

- $q_*(a)$: The **true value** (expected reward) of taking action a.
- R_t: The **reward** received at time t.
- $A_t = a$: The condition that the **action chosen** at time t is a.
- $\mathbb{E}[\cdot]$: Expectation — the **average** over many trials.

K-armed Bandit Problem

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

- A_t - action selected on time step t
- Q_t (a) - estimated value of action a at time step t
- q_*(a) - value of an arbitrary action a

Note: If you knew the value of each action, then it would be trivial to solve the k -armed bandit problem: you would always select the action with highest value :-)

K-armed Bandit Problem

$$\mathbb{E}[a] = -\$0.5$$

$$\mathbb{E}[b] = -\$0.2$$

$$\mathbb{E}[c] = \$0.1$$

$$\mathbb{E}[d] = \$0.11$$

K-armed Bandit Problem

-1, -1, 5

$\hat{E}[a] = 1$

-0.2, -0.2

$\hat{E}[b] = -0.2$

-0.5, -0.5, -0.5

$\hat{E}[c] = -0.5$

-2, -2

$\hat{E}[d] = -2$

Keep pulling the levers; update the estimate of action values;

K-armed Bandit Problem

-1, -1, 5

$\hat{\mathbb{E}}[a] = 1$

-0.2, -0.2

$\hat{\mathbb{E}}[b] = -0.2$

-0.5, -0.5, -0.5

$\hat{\mathbb{E}}[c] = -0.5$

-2, -2

$\hat{\mathbb{E}}[d] = -2$

K-armed Bandit Problem

1. How to maintain the estimate of expected rewards for each action?

   Average the rewards actually received !!!

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

Suppose we have the following sequence of rewards and actions:

Time i	Action A_i	Reward R_i
1	a_1	3
2	a_2	5
3	a_1	4
4	a_1	2

To estimate the average reward for action a_1 at $t = 5$:

$$\sum_{i=1}^{4} R_i \cdot 1_{A_i=a_1} = 3 + 4 + 2 = 9$$

$$\sum_{i=1}^{4} 1_{A_i=a_1} = 3$$

$$\text{Average reward for } a_1 = \frac{9}{3} = 3$$

This tells us that the estimated value for action a_1 is 3.

1. How to use the estimate in selecting the right action?

   Greedy Action Selection

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

K-armed Bandit Problem

2. How to use the estimate in selecting the right action?

   Greedy Action Selection

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

Actions which are inferior by the value estimate upto time t, could be indeed better than the greedy action at t !!!

3. Exploration vs. Exploitation?

   ε-Greedy Action Selection / near-greedy action selection

   Behave greedily most of the time; Once in a while, with small probability **ε** select randomly from among all the actions with equal probability, independently of the action-value estimates.

K-armed Bandit Problem

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

K-armed Bandit Problem

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

1. First machine — action a

- Rewards observed so far: `{-1, -1, 5}`
- $N_{11}(a) = 3$ → You've pulled this arm **3 times**.
- $q_*(a) = -\$0.5$ → The **true expected reward** (the "real" value hidden from the agent).
  You can't see this — it's just for illustration. On average, this machine gives **−$0.5 per play**.
- $Q_{11}(a) = 1$ → The **estimated value** (your current guess based on experience).

So: even though the true mean reward is −$0.5, your estimate after seeing {−1, −1, 5} is **+1** (the sample average).

$$Q_{11}(a) = \frac{(-1) + (-1) + 5}{3} = 1$$

K-armed Bandit Problem

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

2. Second machine — action b

- Rewards: `{-0.2, -0.2}`
- $N_{11}(b) = 2$
- $q_*(b) = -\$0.2$
- $Q_{11}(b) = -0.2$

Here, your estimate **matches the true average**, because the rewards are consistent.

$$Q_{11}(b) = \frac{(-0.2) + (-0.2)}{2} = -0.2$$

K-armed Bandit Problem

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

🎰 3. Third machine — action c

- Rewards: `{-0.5, -0.5, -0.5}`
- $N_{11}(c) = 3$
- $q_*(c) = +\$0.1$
- $Q_{11}(c) = -0.5$

Here, the true value is **positive**, but your experience so far was unlucky — all draws were −0.5 — so your current estimate $Q_{11}(c) = -0.5$ is **too pessimistic**.

K-armed Bandit Problem

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

4. Fourth machine — action d

- Rewards: `{-2, -2}`
- $N_{11}(d) = 2$
- $q_*(d) = +\$1$
- $Q_{11}(d) = -2$

This one is striking:

- The **true value** $q_*(d) = +1$: actually a *great* machine.
- But so far, the agent got two unlucky plays giving −2 each time → $Q_{11}(d) = -2$.
- Because of those bad early samples, the agent currently **believes it's terrible** — even though it's the best arm!

K-armed Bandit Problem

Greedy Action

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

K-armed Bandit Problem

Action to Explore

{-1, -1, 5}

$N_{11}(a)=3$

$q_*(a)=-\$0.5$

$Q_{11}(a)=1$

{-0.2, -0.2}

$N_{11}(b)=2$

$q_*(b)=-\$0.2$

$Q_{11}(b)=-0.2$

{-0.5, -0.5, -0.5}

$N_{11}(c)=3$

$q_*(c)=\$0.1$

$Q_{11}(c)=-0.5$

{-2, -2}

$N_{11}(d)=2$

$q_*(d)=\$1$

$Q_{11}(d)=-2$

K-armed Bandit Problem

ε-Greedy Action Selection / near-greedy action selection

```
epsilon = 0.05 // small value to control exploration
def get_action():
    if random.random() > epsilon:
        return argmaxa(Q(a))
    else:
        return random.choice(A)
```

This code snippet implements the **ε-greedy (epsilon-greedy) action selection strategy**, commonly used in **Reinforcement Learning** especially in **multi-armed bandit** and **Q-learning** problems.

K-armed Bandit Problem

ε-Greedy Action Selection / near-greedy action selection

```
epsilon = 0.05 // small value to control exploration
def get_action():
    if random.random() > epsilon:
        return argmaxa(Q(a))
    else:
        return random.choice(A)
```

This code snippet implements the **ε-greedy (epsilon-greedy) action selection strategy**, commonly used in **Reinforcement Learning** especially in **multi-armed bandit** and **Q-learning** problems.

1. `epsilon = 0.05`
   - Defines the **exploration rate** (ε).
   - With probability **0.05 (5%)**, the agent will **explore** (try a random action).
   - With probability **0.95 (1 – ε)**, it will **exploit** (choose the best-known action).
2. `if random.random() > epsilon:`
   - `random.random()` generates a number between 0 and 1.
   - If it's **greater than 0.05**, the agent **exploits** (chooses the best-known action).
3. `return argmaxa(Q(a))`
   - Selects the **action** `a` that has the **highest estimated Q-value** (the one currently believed to give the most reward).
   - This is the **greedy choice** — the best action according to current knowledge.
4. `else: return random.choice(A)`
   - Otherwise, pick a **random action** from the set of all actions **A**.
   - This ensures the agent still **explores new possibilities**.

K-armed Bandit Problem

ε-Greedy Action Selection / near-greedy action selection

```
epsilon = 0.05 // small value to control exploration
def get_action():
    if random.random() > epsilon:
        return argmaxa(Q(a))
    else:
        return random.choice(A)
```

You're playing a **3-armed bandit** game — meaning you have **3 actions (A1, A2, A3)**.

You've estimated the current **Q-values** (expected rewards) for each arm as:

Action	Q(a) (estimated value)
A1	2.0
A2	1.5
A3	0.5

Trial	Random number	Decision	Action chosen
1	0.07	Explore	Random (A3)
2	0.64	Exploit	Best (A1)
3	0.92	Exploit	Best (A1)
4	0.03	Explore	Random (A2)

Epsilon-greedy setup

$$\varepsilon = 0.1$$

10% chance → **explore (random action)**

90% chance → **exploit (best-known action)**

K-armed Bandit Problem

ε-Greedy Action Selection / near-greedy action selection

```
epsilon = 0.05 // small value to control exploration
def get_action():
    if random.random() > epsilon:
        return argmaxa(Q(a))
    else:
        return random.choice(A)
```

- **In the limit as the number of steps increases, every action will be sampled by ε-greedy action selection an infinite number of times. This ensures that all the Qt (a) converge to q_* (a).**
- **Easy to implement / optimize for epsilon / yields good results**

Ex-1: In ε-greedy action selection, for the case of two actions and ε = 0.5, what is the probability that *the greedy action* is selected?

Ex-1: In ε-greedy action selection, for the case of two actions and ε = 0.5, what is the probability that *the greedy action* is selected?

Recall - What does ε-greedy mean?

- With probability (1 − ε) → the agent exploits (chooses the greedy/best action).
- With probability ε → the agent explores (chooses randomly among all actions).

So, there are two possible modes of selection:

1. Greedy selection (prob = 1 − ε)
2. Random selection (prob = ε)

Ex-1: In ε-greedy action selection, for the case of two actions and ε = 0.5, what is the probability that *the greedy action* is selected?

p (greedy action)

= p (greedy action AND greedy selection) + p (greedy action AND random selection)

Here, Use the conditional probability rule, $p(A \text{ and } B) = p(A \mid B)p(B)$:

=p (greedy action | greedy selection) p (greedy selection)

+ p (greedy action | random selection) p (random selection)

= p (greedy action | greedy selection) (1-**ε**) + p (greedy action | random selection) (**ε**)

= p (greedy action | greedy selection) (0.5) + p (greedy action | random selection) (0.5)

Here, $p(\textit{greedy action} \mid \textit{greedy selection}) = 1$) If we are in greedy mode, we *always* pick the greedy action.)

$p(\textit{greedy action} \mid \textit{random selection}) = \frac{1}{2}$ If we choose randomly among 2 actions, there's a 50% chance we pick the greedy one.)

= (1) (0.5) + (0.5) (0.5)

= 0.5 + 0.25

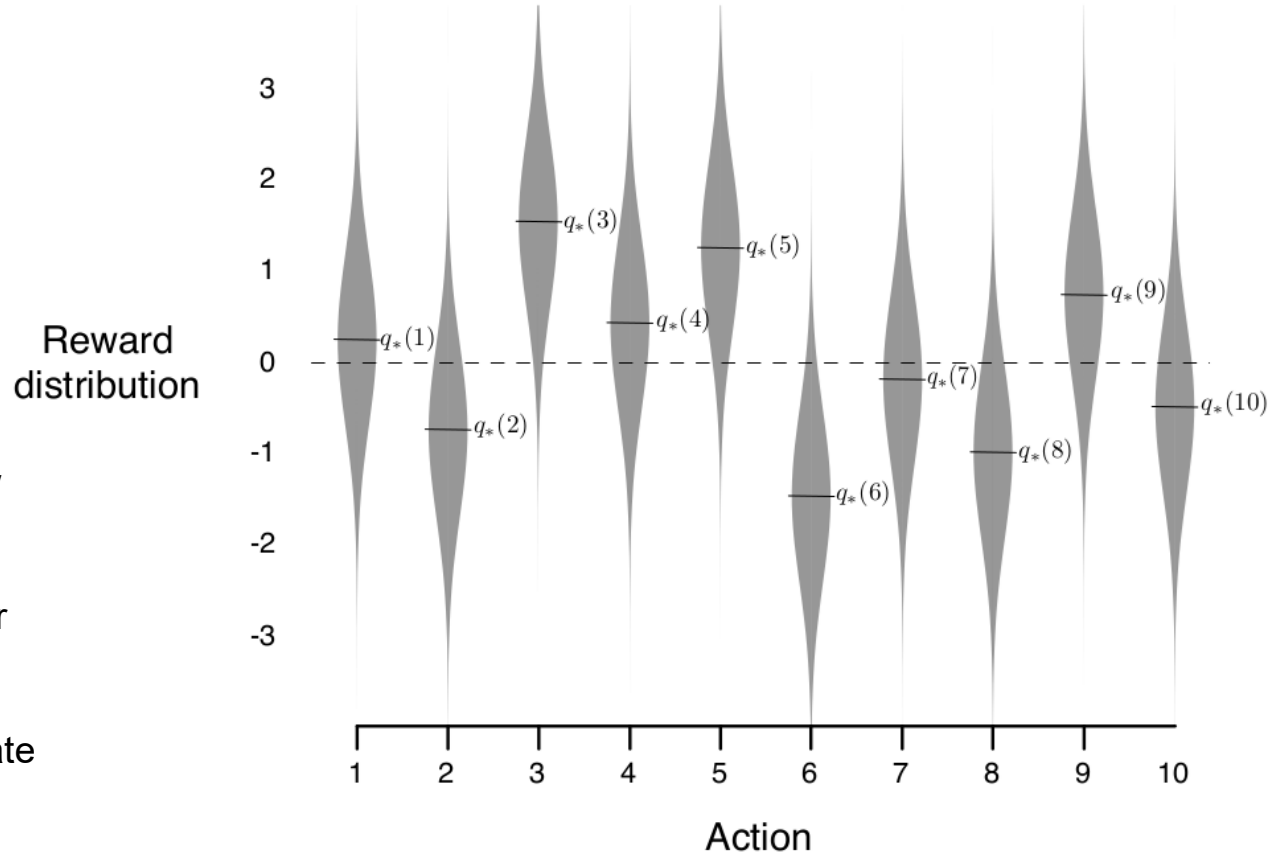
= 0.75

10-armed Testbed

Imagine you are playing a slot machine with **10 levers (arms)**.

- Each lever gives a **random reward** every time you pull it.
- Some levers are better *on average* than others — but you don't know which!
- Your goal: **maximize total reward** by learning which arm is best over time.

This is a **simplified Reinforcement Learning problem** — there's no state or environment, just *actions* and *rewards*.

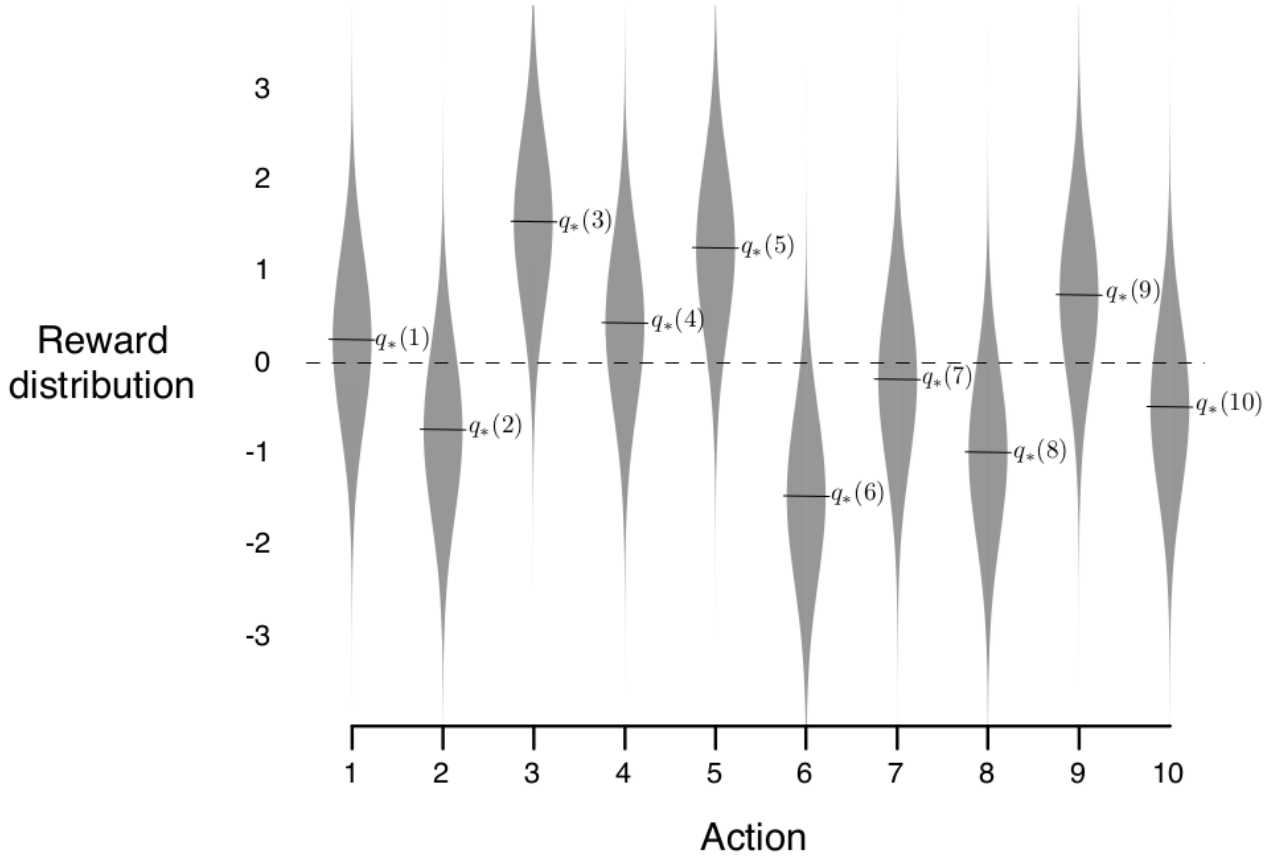


An example bandit problem from the 10-armed testbed

10-armed Testbed

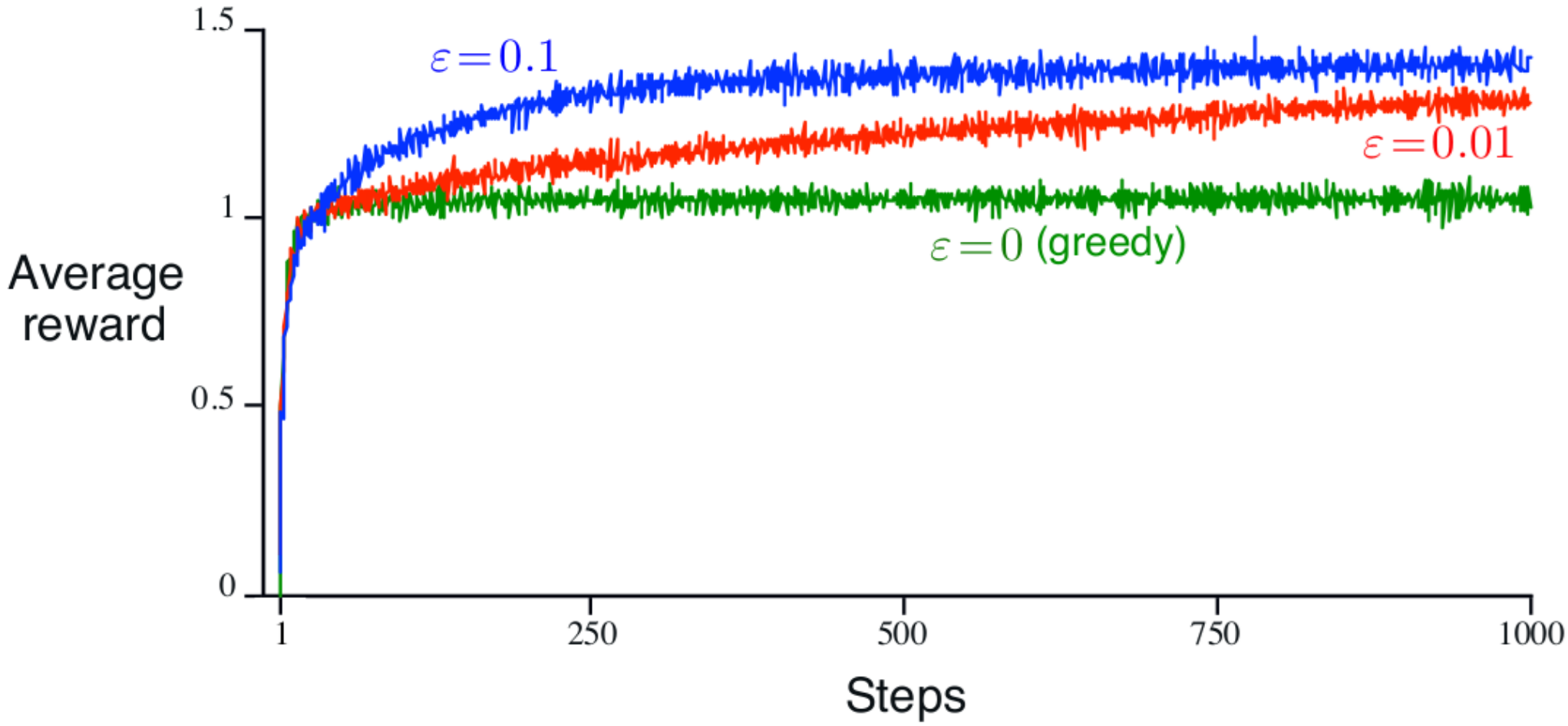
Example:

- A set of 2000 randomly generated k -armed bandit problems with k = 10
- Action values were selected according to a normal (Gaussian) distribution with mean 0 and variance 1.
- While selecting action A_t at time step t, the actual reward, R_t , was selected from a normal distribution with mean $q_*(At)$ and variance 1
- One Run : Apply a method for 1000 time steps to one of the bandit problems
- Perform 2000 runs, each run with a different bandit problem, to get an algorithms average behavior

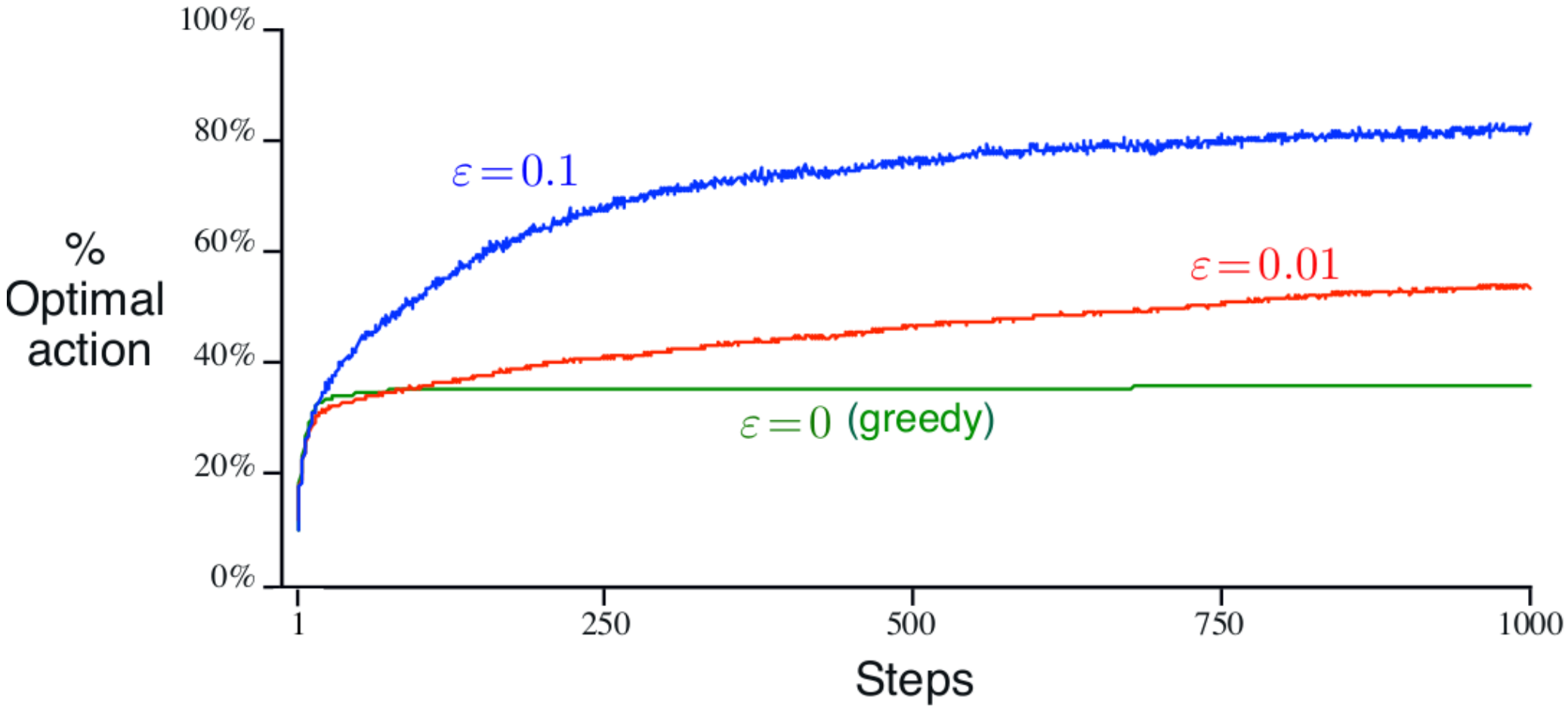


An example bandit problem from the 10-armed testbed

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



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



Discussion on Exploration vs. Exploitation

1) What if the reward variance is
    a. larger, say 10 instead of 1?
    b. zero ? [deterministic]

2) What if the bandit task is non-stationary? [that is, the true values of the actions changed over time]

Discussion on Exploration vs. Exploitation

1) What if the reward variance is
   a. larger, say 10 instead of 1?
   b. zero ? [deterministic]

2) What if the bandit task is non-stationary? [that is, the true values of the actions changed over time]

1)

Larger reward variance (e.g., 10 instead of 1):

- Rewards become **noisier** → harder to estimate true action values.
- Learning becomes **slower** and requires **more samples** to converge.

If variance = 0 (deterministic):

- Rewards are always the same → learning is **instant and exact** after one trial.

Discussion on Exploration vs. Exploitation

1) What if the reward variance is
   a. larger, say 10 instead of 1?
   b. zero ? [deterministic]

2) What if the bandit task is non-stationary? [that is, the true values of the actions changed over time]

2)

Non-stationary bandit (true values change over time):

- Past knowledge becomes outdated.
- The agent must **adapt continuously**

Ex-2:

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 Q1 (a) = 0, for all a.

Suppose the initial sequence of actions and rewards is A1 = 1, R1 = 1, A2 = 2, R2 = 1, A3 = 2, R3 = 2, A4 = 2, R4 = 2, A5 = 3, R5 = 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?

NOTE:

The naive way to estimate the true value of an action a, we take the **sample average** of all rewards obtained from that action up to time n:

$$Q_n = \frac{R_1 + R_2 + \cdots + R_n}{n}$$

- But this means you must remember **all past rewards** $R_1, R_2, \ldots, R_n$.
  That's **inefficient** in terms of memory and computation.

SO, The incremental (efficient) way:

We can compute the **new average** using only:

- the **old average** (Q_n)
- the **new reward** (R_n)

and the **count of observations** (n).

This avoids storing all the data.

Incremental Implementation

- It's an **efficient way to update the estimated value of an action** (its average reward) **incrementally**, *without storing all past rewards.*

  - Efficient approach to compute the estimate of action-value;

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

  - Given Qn and the nth reward, Rn , the new average of all n rewards can be computed as follows

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

The **new estimate** = old estimate + (step size) × (error).

$R_n - Q_n$ → **error term** (difference between actual reward and estimated value)

$\frac{1}{n}$ → **step size** (gets smaller as we collect more data)

Q_{n+1} → **updated estimate** of the true value

Incremental Implementation

Note:

- StepSize decreases with each update
- We use α or $\alpha_t(a)$ to denote step size (constant / varies with each step)

Discussion:

Const vs. Variable step size?

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

$$NewEstimate \leftarrow OldEstimate + StepSize\Big[Target - OldEstimate\Big]$$

Bandit Algorithm with Incremental Update/ ε-greedy selection

Initialize, for $a = 1$ to k:

$$Q(a) \leftarrow 0$$
$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$$R \leftarrow bandit(A)$$
$$N(A) \leftarrow N(A) + 1$$
$$Q(A) \leftarrow Q(A) + \tfrac{1}{N(A)}\left[R - Q(A)\right]$$

k	Number of possible actions (arms) in the bandit problem.
Q(a)	Estimated value (average reward) of action *a*.
N(a)	Number of times action *a* has been selected so far.
ε (epsilon)	Probability of exploring (choosing a random action).
A	The action chosen at the current time step.
argmax_a Q(a)	The action *a* with the highest estimated value (greedy choice).
bandit(A)	Function that returns the actual reward *R* for taking action *A*.
R	The reward received after selecting action *A*.

What are problems whose solutions are modelled as MAB?

Common Problems Modeled as MAB

Domain	Example Problem	Why it's a Bandit Problem
📰 **Online Advertising**	Choosing which ad to display to a user	Each ad (arm) gives an uncertain click reward (CTR). The system must balance trying new ads vs. showing the best one.
🛍️ **Product Recommendation**	Recommending products to users on Amazon	Each recommendation can lead to a purchase (reward). The agent learns which items perform best.
📱 **News / Content Personalization**	Showing articles or videos to users	Each content piece gives engagement (clicks, watch time). Need to explore new topics and exploit known favorites.
💰 **Investment / Portfolio Selection**	Choosing among different stocks or assets	Each stock (arm) yields uncertain returns. The goal is to learn which gives the best long-term reward.
🎮 **A/B Testing / Website Optimization**	Testing design A vs. B (or multiple variants)	Each version has a conversion rate; MAB learns which version performs best faster than standard A/B testing.
🚚 **Route Selection / Traffic Control**	Choosing paths for vehicles or signals at intersections	Each route or signal timing yields uncertain travel times or throughput.
🧠 **Clinical Trials**	Selecting which treatment to give to patients	Each treatment (arm) has an unknown success rate. MAB can balance exploring new drugs vs. using known effective ones.
🎓 **Education / e-Learning**	Selecting next learning material for a student	Each activity's effect on learning (reward) is uncertain; the system adapts to maximize learning gains.
🔍 **Search Engine Optimization**	Deciding which result or snippet to display	Click feedback from users helps refine which items are more rewarding (relevant).

Non-stationary Problem

- Most RL problems are non-stationary !
- Give more weight to recent rewards than to long-past rewards !!!

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

- **Stationary:** "Learn once, stays true."
- **Non-stationary:** "Keep learning — the world changes!"

Non-stationary Problem

- Most RL problems are non-stationary !
- Give more weight to recent rewards than to long-past rewards !!!

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

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

Exponential recency-weighted average (ERWA)

It's a method to estimate the **average reward** of an action while **giving more weight to recent rewards** and **less weight to older ones.** It's useful for **non-stationary problems** (where true values change over time).

→ Each past reward R_i is weighted by $(1-\alpha)^{n-i}$,
meaning its influence **decays exponentially** as it gets older.

Optimistic Initial Values

- All the above discussed methods are ***biased*** by their initial estimates
- For sample average method the bias disappears once all actions have been selected at least once
- For methods with constant α, the bias is permanent, though decreasing over time
- Initial action values can also be used as a simple way of encouraging exploration.
- In 10 armed testbed, set initial estimate to +5 rather than 0.

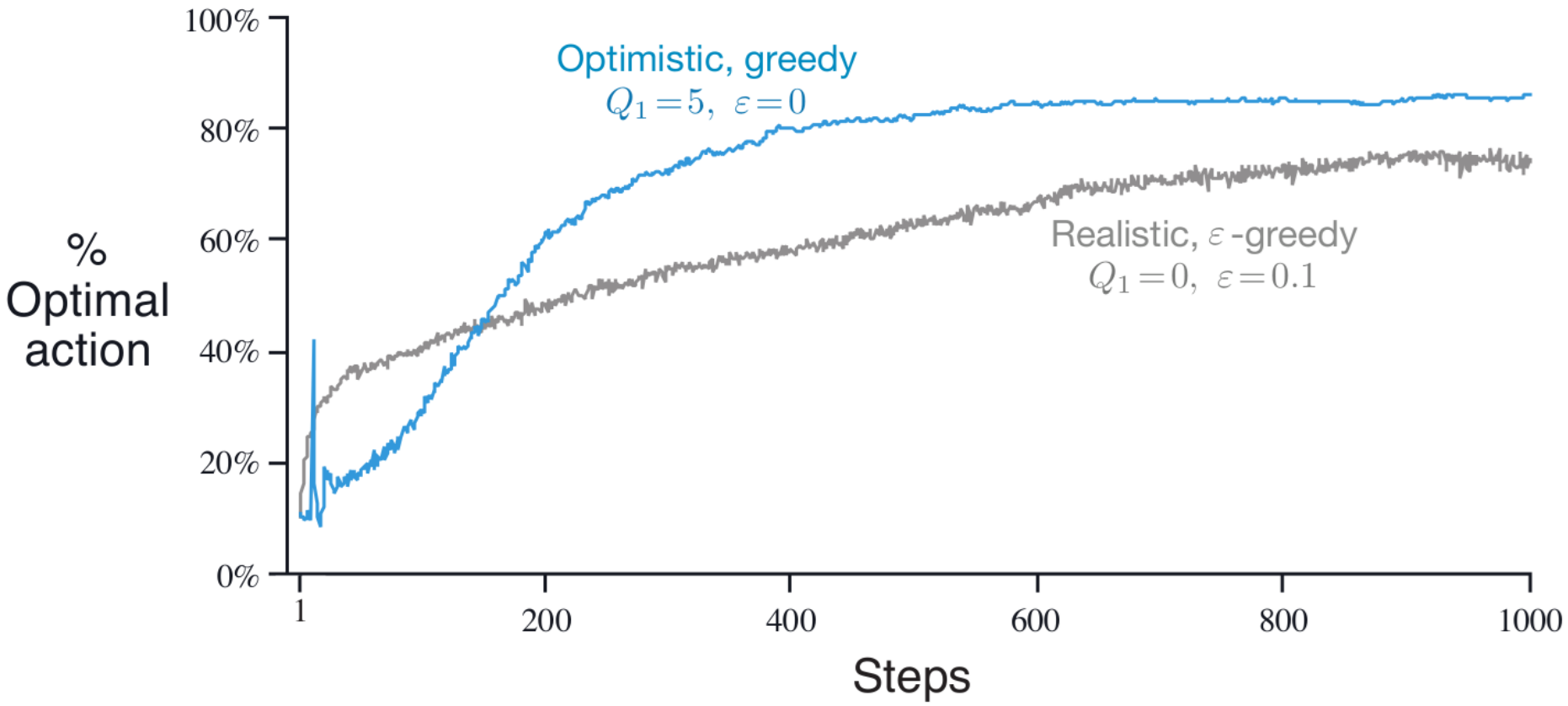
  This can encourage action-value methods to explore.

  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.

Optimistic Initial Values



Caution:
Optimistic Initial Values can only be considered 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.

Question:
Explain how in the non-stationary scenario the optimistic initial values will fail (to explore adequately).

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

Upper-Confidence-Bound Action Selection

- **ε**-greedy action selection forces the non-greedy actions to be tried,
  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
  Take into account both how close their estimates are to being maximal and the uncertainties in those estimates.

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

Upper-Confidence-Bound Action Selection

- Each time a is selected the uncertainty is presumably reduced
- 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.
- Actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time

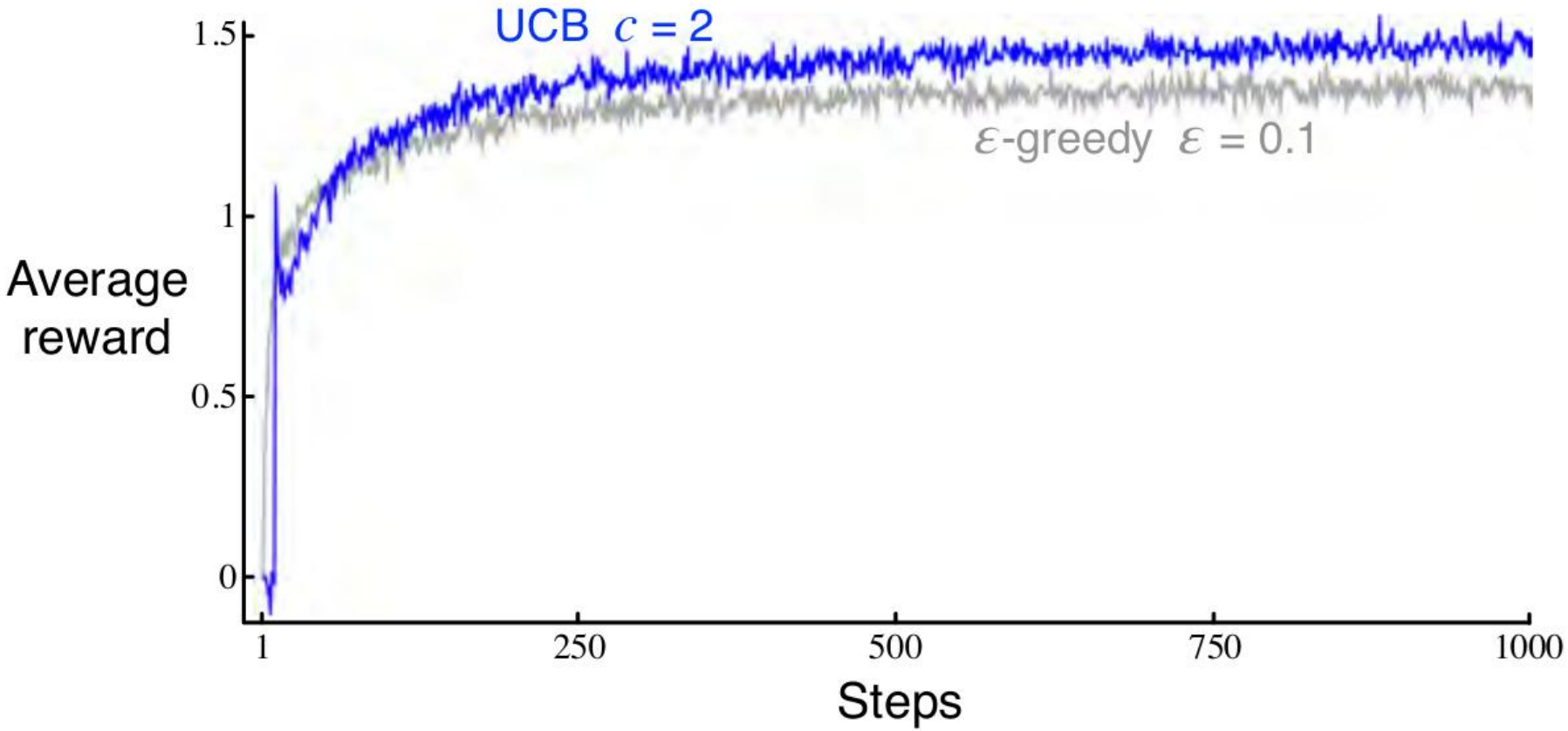
Action Value at time t for a

Confidence Level

Measure of Uncertainty

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

Upper-Confidence-Bound Action Selection



UCB often performs well, as shown here, but is more difficult than "-greedy to extend beyond bandits to the more general reinforcement learning settings

Policy-based algorithms

- Forget about action-value (Q) estimates, we don't really care about them
- We care about what actions to chose

Let's assign a preference to each action and tweak its value

- Define $H_t(a)$ as a numerical preference value associated with action a
- Which action is selected?
  - $A_t = \underset{a}{\text{argmax}}[H_t(a)]$
    - Hardmax results in no exploration -- deterministic action selection

Softmax!

Softmax function

- **Input**: vector of preferences
- **Output**: vector of probabilities forming a valid distribution
- $\Pr\{a_t = a\} = \frac{e^{H_t(a)}}{\sum_{a' \in A} e^{H_t(a')}} = \Pr(a)$
- $H_t \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} 6 \\ 9 \\ 2 \end{bmatrix}$
- $\text{softmax} \begin{pmatrix} 6 \\ 9 \\ 2 \end{pmatrix} = \begin{bmatrix} 0.047 \\ 0.952 \\ 0.00087 \end{bmatrix}$
- That is, with $\Pr(0.95)$ choose a_2, $\Pr(0.05)$ choose a_1, and $< \Pr(0.01)$ choose a_3

Softmax function

- Exploration – checked!
- Softmax provides another important attribute – **a differentiable policy**
- Say that we learn that action a_1 results in good relative performance
- Hardmax: $A_t = \underset{a}{\text{argmax}}[H_t(a_1), H_t(a_2), H_t(a_3)]$
  - Change $H(a_1)$ such that $\Pr(a_1)$ is increased, $\frac{\partial \Pr(a_1)}{\partial H(a_1)} = NA$
- Softmax: $\Pr(a_1) = \frac{e^{H_t(a)}}{\sum_{a' \in A} e^{H_t(a')}}$
  - Change $H(a_1)$ such that $\Pr(a_1)$ is increased -> update towards $\frac{\partial \Pr(a_1)}{\partial H(a_1)}$

Gradient ascend

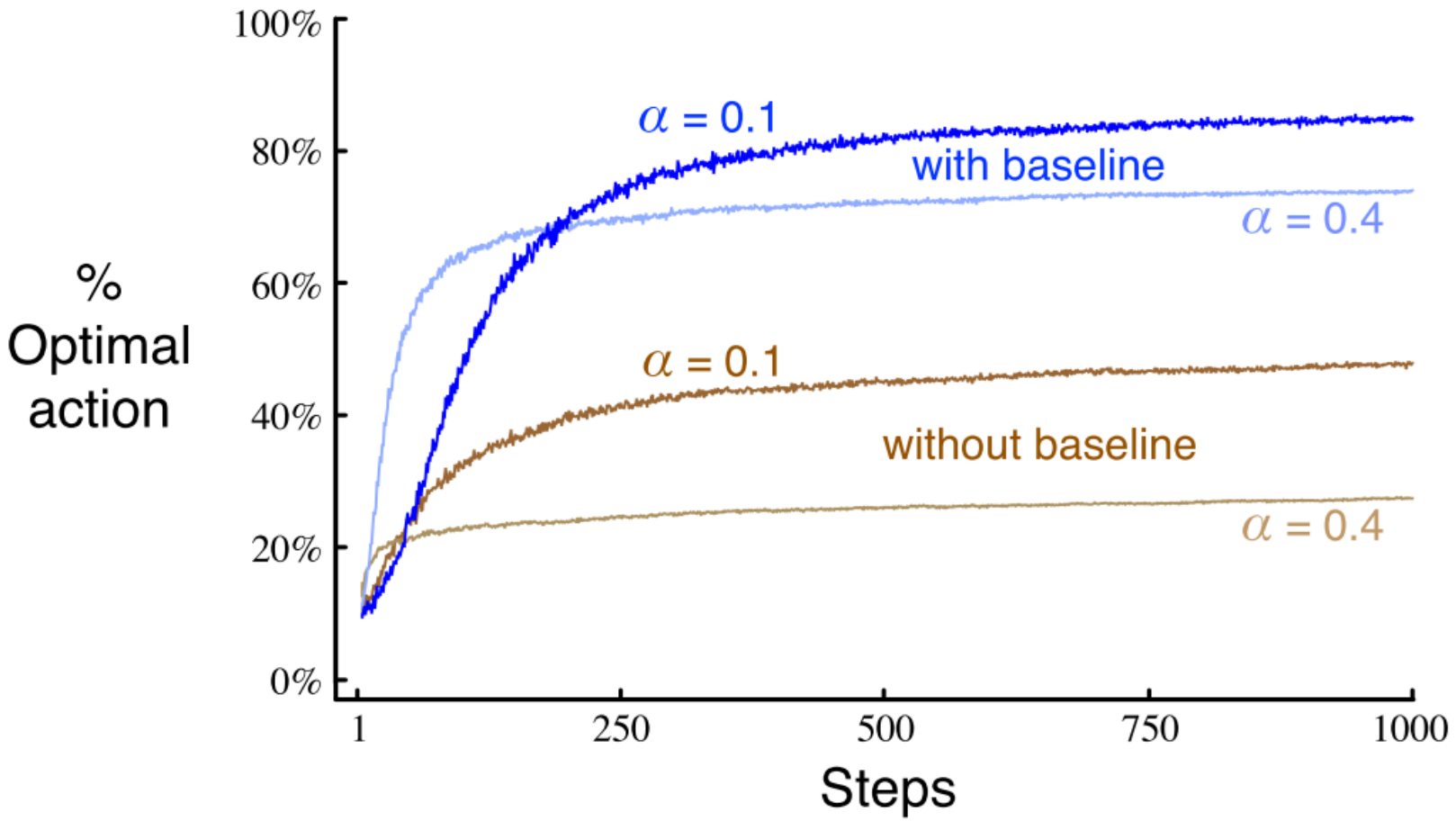
- $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})\frac{\partial \Pr(A_t)}{\partial H(A_t)}$?
- $\forall a \neq A_t, H_{t+1}(a) = H_t(a) + \alpha(R_t - \overline{R_t})\frac{\partial \Pr(A_t)}{\partial H(a)}$
- Update the preferences based on observed reward and a baseline reward ($\overline{R_t}$)
- If the observed reward is larger than the baseline:
  - Increase the preference of the chosen action, A_t
  - Decrease the preference of all other actions, $\forall a \neq A_t$
- Else do the opposite

Update Rule

On each step, after selecting action A t and receiving the reward Rt ,
Update the action preferences :

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \Pr(A_t))$$
$$\forall a \neq A_t, H_{t+1}(a) = H_t(a) - \alpha(R_t - \overline{R_t}) \Pr(a)$$

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

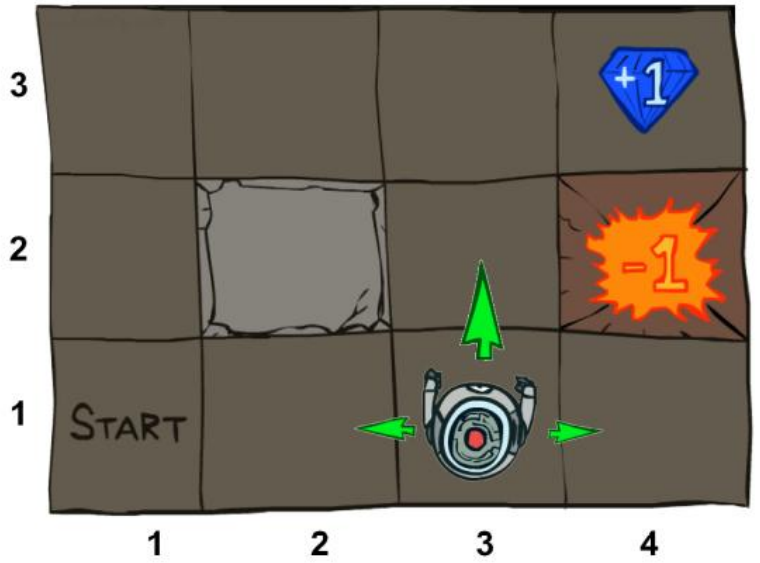

% Optimal action
100%
80%
60%
40%
20%
0%
1
250
500
750
1000
Steps
α = 0.1
with baseline
α = 0.4
α = 0.1
without baseline
α = 0.4

What did we learn?

- **Problem**: choose the action that results in highest expected reward
- **Assumptions**: 1. actions' expected reward is unknown, 2. we are confronted with the same problem over and over, 3. we are able to observe an action's outcome once chosen
- **Approach**: learn the actions' expected reward through exploration (value based) or learn a policy directly (policy based), exploit learnt knowledge to choose best action
- **Methods:** 1. greedy + initializing estimates optimistically, 2. epsilon-greedy, 3. Upper-Confidence-Bounds, 4. gradient ascend + soft-max

A different scenario

- Associative vs. Non-associative tasks ?
- Policy: A mapping from situations to the actions that are best in those situations
- (discuss) How do we extend the solution for non-associative task to an associative task?
  - **Approach**: Extend the solutions to non-stationary task to non-associative tasks
    - Works, if the true action values changes slowly
  - What if the context switching between the situations are made explicit?
    - How?
    - Need Special approaches !!!



Required Readings

1. Chapter-2 of Introduction to Reinforcement Learning,2nd Ed., Sutton & Barto
2. A Survey on Practical Applications of Multi-Armed and Contextual Bandits, Djallel Bouneffouf , Irina Rish [https://arxiv.org/pdf/1904.10040.pdf]

Thank you !