Multi-arm Bandits

The simplest reinforcement learning problem

Intro

RL uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions

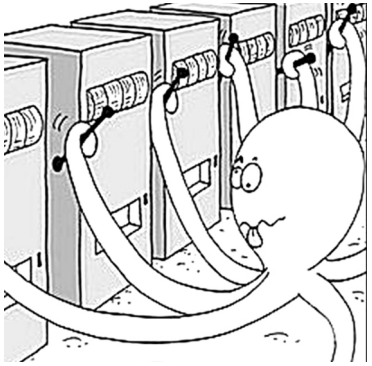
- need for active exploration

RL Vs Supervised learning: *evaluative* feedback (RL) depends entirely on the action taken, whereas *instructive* feedback (SL) is independent of the action taken

Start from *nonassociative* setting

- Learning to act in one situation
- Simple version of the *k*-armed bandit problem

k-Armed Bandit Problem



Choice repeatedly among *k* different options, or actions

After each choice receive a numerical reward from a stationary probability distribution that depends on the action you selected

Objective: maximize the *expected total reward* over some time period
e.g. over 1000 action selections (*time steps)*

How to learn?
Try & pull each arm many times

k-Armed Bandit Problem

Each of the *k* actions has an expected reward given that the action is selected - optimal value of the action $q_*(a)$ $q_*(a) = E\left[R_t \middle| A_t = a\right]$

where A_t – the action selected on time step *t* and R_t – the corresponding reward.

Compute the expected reward when action a is taken. Example:

Action (A_t)	Reward (R_t)			
a_1	5	7	6	8
a_2	4	6	5	7

$$E[R_t \mid A_t = a_1] = \frac{5+7+6+8}{4} = \frac{26}{4} = 6.5$$

$$E[R_t \mid A_t = a_2] = \frac{4+6+5+7}{4} = \frac{22}{4} = 5.5$$

Identify Optimal Action $q_*(a_1)=6.5$ $q_*(a_2)=5.5$

Thus, the optimal action is a1, as it has the highest expected reward.

Assumption: we do not know the action values with certainty. Otherwise, just select the action with highest value

Denote the estimated value of action *a* at time step *t* as $Q_t(a)$ .We would like $Q_t(a)$ to be close to $q_*(a)$

Action-value Methods

Recall: the true value of an action is the mean reward when that action is selected $q_*(a)$

Natural way to estimate by averaging the rewards *actually* received:

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

Sample-average method for estimating action values - Not necessary the best way of estimation

Incremental Implementation

How the sample averages for action values estimation can be computed in a computationally efficient manner?

Recursive average
Suppose there is a *single* action
R_i - the reward received after the i-th selection *of this action*
Q_n - the estimate of the action value after it has been selected n - 1 times:

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

Updating average:

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

requires memory only for Q_n and n and simple computation

Incremental Implementation

$$x_1 + x_2 + \cdots + x_N = N \cdot A_N$$

$$A_{N+1} = \frac{N \cdot A_N + x_{N+1}}{N+1}$$

$$K = \frac{1}{N+1}$$

$$\frac{N}{N+1} = 1 - K$$

$$A_{N+1} = \frac{N \cdot A_N}{N+1} + \frac{x_{N+1}}{N+1}$$

$$A_{N+1} = A_N \cdot (1-K) + x_{N+1} \cdot K$$

$$A_{N+1} = A_N \cdot \frac{N}{N+1} + x_{N+1} \cdot \frac{1}{N+1}$$

$$A_{N+1} = A_N + K(x_{N+1} - A_N)$$

Greedy Action Selection Method

Simplest action selection rule: select one of the actions with the highest estimated value - one of the *greedy* actions

greedy action selection method

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

Exploits current knowledge to maximize immediate reward

Agent interacting with an environment: actions a_1, a_2, a_3.
Initial Estimates: $Q_0(a_1)=5$ $Q_0(a_2)=7$ $Q_0(a_3)=6$ (initialized randomly, based on prior knowledge)
Step 1: Action with the Highest Value, t = 0, $A_t = \text{argmax}_a\, Q_t(a) = a_2$
Step 2: Observe the Reward: take action a_2, observes a reward R_t, assume: $R_t = 8$
Step 3: Update the Action Value $Q_{t+1}(a) = Q_t(a) + \alpha \cdot [R_t - Q_t(a)]$

$$Q_1(a_2) = 7 + 0.1 \cdot [8 - 7] = 7 + 0.1 \cdot 1 = 7.1$$

Updated estimates: $Q_1(a_1) = 5$(unchanged) $Q_1(a_2) = 7.1$(updated) $Q_1(a_3) = 6$(unchanged)
Step 4: Repeat the Process, t = 1, $A_t = \text{argmax}_a\, Q_t(a) = a_2$, agent selects a_2 again.
Step 5: Continue Observing and Updating
Over time, the estimates Q(a) converge to the true values $q_*(a)$, assuming the rewards are sampled from a stationary distribution.

Sample-Average : If we used the sample-average method, the agent would give equal weight to all past rewards, making it slower to adapt to changes.

Tradeoff :

- A smaller α makes the agent more stable but slower to adapt.
- A larger α makes the agent more responsive but potentially noisier.

Greedy Behavior:
- The agent always selects the action with the highest estimated value.
- This maximizes immediate rewards but ignores exploration, potentially missing better actions if the estimates are inaccurate.

Limitation: Purely greedy action selection can get stuck in suboptimal actions if the initial estimates are misleading or if the environment is stochastic.

GreedyActionSelection

Inputs:
Q: A dictionary or array storing the estimated action values (Q(a)).
Actions: A list of all possible actions.

Initialize variables to track the best action and its value
best_action = None
best_value = start with a very small value

Loop through all actions
Get the estimated value of the current action value = Q[action]

Find action that has a higher value than the current best

Return the action with the highest estimated value

```
def greedy_action_selection(Q, actions):
    """
    Selects the action with the highest estimated value (Q(a)) using the Greedy Action Selection Method.
    Parameters:
    - Q (dict): A dictionary where keys are actions and values are their estimated values (Q(a)).
    - actions (list): A list of all possible actions.
      Returns:
    - best_action: The action with the highest estimated value.
    """
    # Initialize variables to track the best action and its value
    best_action = None
    best_value = float('-inf')  # Start with a very small value

    # Loop through all actions
    for action in actions:
        # Get the estimated value of the current action
        value = Q.get(action, 0)  # Default to 0 if action is not in Q

        # Check if this action has a higher value than the current best
        if value > best_value:
            best_value = value
            best_action = action

    # Return the action with the highest estimated value
    return best_action


# Example usage
if __name__ == "__main__":
    # Estimated action values (Q(a))
    Q = {"a1": 5, "a2": 7, "a3": 6}

    # List of possible actions
    actions = ["a1", "a2", "a3"]

    # Select the greedy action
    best_action = greedy_action_selection(Q, actions)

    # Print the result
    print(f"The best action is: {best_action}")
```

Exploration Vs Exploitation

At time t there is an action with greatest estimated value: *greedy* action

Exploitation: selection one of greedy actions $A_t = A_t^*$

- the right thing to do to maximize the expected reward on the *one step*

Exploration: selection one of the nongreedy actions $A_t \neq A_t^*$

- enables to improve estimate of the nongreedy action's value
- may produce the greater total reward in the *long run*

Possible uncertainty: at least one of nongreedy actions probably is actually better than the greedy action but you don't know which one

Regret

The *action-value* is the mean reward for action a: $q^*(a)=\mathrm{E}[r|a]$

The *optimal value* V^* is $V^* = Q(a*) = \max q^*(a)$ (for $a \in \mathrm{A}$)

The *regret* is the opportunity loss for one step $l_t = \mathrm{E}\,[V^* - Q(a_t)]$

The *total regret* is the total opportunity loss

$$L_t = \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right]$$

Minimize regret = maximize total reward

Action (a)	Reward Distribution (r)	True Mean Reward ($q_*(a)$)
a_1	N(5,1)	$q_*(a_1) = 5$
a_2	N(7,1)	$q_*(a_2) = 7$
a_3	N(6,1)	$q^*(a_3) = 6$

V_*=7(corresponding to action a_2)
Suppose selects action $a_1$at time t
l_t=7−5=2
If the agent had selected
the optimal action a_2,
the regret would be: l_t=7−7=0

Compute the Optimal Value (V∗)

$$V^* = \max(q_*(a_1), q_*(a_2), q_*(a_3)) = \max(5, 7, 6) = 7$$

$$l_t = E[V^* - Q(a_t)] = V^* - q_*(a_t)$$

Time Step(t)	Action Taken (a_t)	True Mean Reward ($q_*(a_t)$)	Regret (l_t)
1	a_1	5	7 - 5 = 2
2	a_2	7	7 - 7 = 0
3	a_1	5	7 - 5 = 2
4	a_3	6	7 - 6 = 1
5	a_2	7	7 - 7 = 0
6	a_1	5	7 - 5 = 2
7	a_2	7	7 - 7 = 0
8	a_3	6	7 - 6 = 1
9	a_2	7	7 - 7 = 0
10	a_1	5	7 - 5 = 2

$$\text{Total Regret} = \sum_{t=1}^{10} l_t = 2 + 0 + 2 + 1 + 0 + 2 + 0 + 1 + 0 + 2 = 10$$

Exploration Vs Exploitation

If there are many time steps ahead:

- may be better to explore the nongreedy actions
- discover which of them are better than the greedy action

Reward is lower in the short run, during exploration

Reward is higher in the long run after you have discovered the better actions, you can exploit *them* many times

Tradeoff between exploration and exploitation

ε-greedy Action Selection Method

The rule: **always** choose the **greedy** action (exploitation) **except** for the relatively small probability of **ε** of choosing a **random** action (exploration)
The *greedy* actions are taken based on which action has the highest expected reward at that particular time step

Tradeoff between exploration & exploitation

- To obtain a lot of reward, agent must prefer (exploit) actions that it has tried in the past (greedy)
- To discover such actions, it has to try(explore) actions that it has not selected before

The sample-average estimates converge to the true values *If* the action is taken an infinite number of times

$$\lim_{N_t(a)\to\infty} Q_t(a) = q_*(a)$$

A Simple Bandit Algorithm

Bandit algorithm with incrementally computed sample averages and ε-greedy action selection

bandit(a) takes an action and return a corresponding reward

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

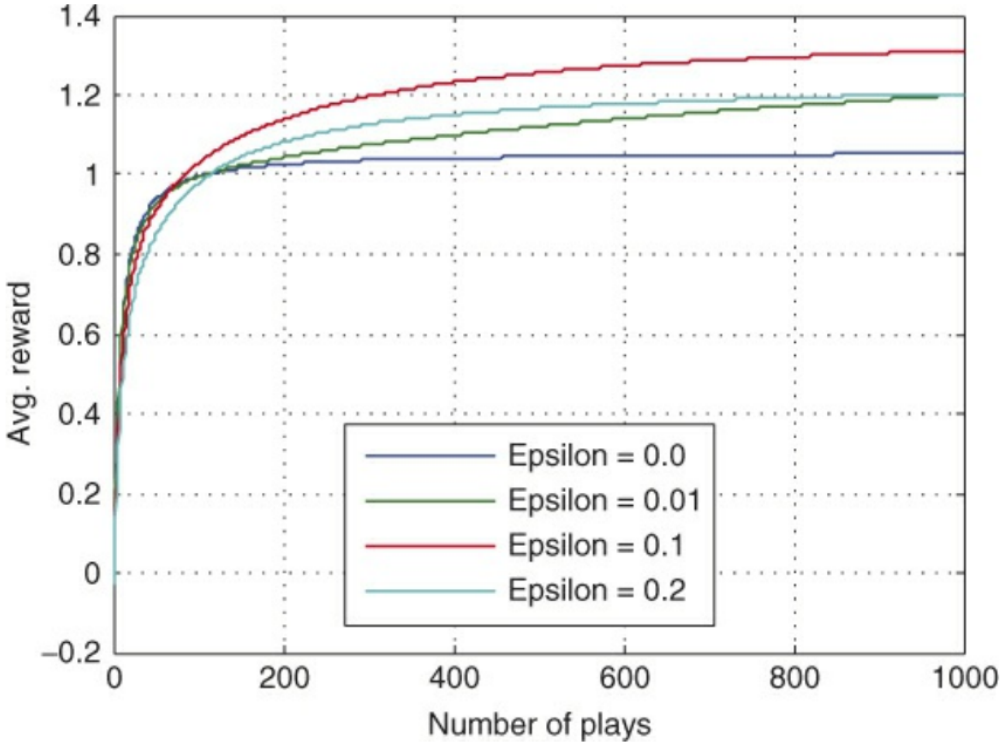
$N(A) \leftarrow N(A) + 1$

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

Example:Python,epsilon_greedy_incremental

Example

Armed bandit with varying ε



Remarks

The advantage of ε-greedy over greedy methods depends on the task

- Large variance (more exploration to find optimal strategy)
- No uncertainty: greedy knows the truth

If the true values of actions changed over time (deterministic case nonstationary task) - exploration is needed to make sure one of the nongreedy actions has not changed to become better than the greedy one

Tracking a Nonstationary Problem

Averaging methods are appropriate for stationary bandit problems
Stationarity: the reward probabilities do not change over time

Idea for nonstationary case: *give more weight to recent rewards than to long-past rewards*

Popular way: use a *constant* step-size parameter

Example

For incremental update rule for averaging: $Q_{n+1} \doteq Q_n + \alpha\left[R_n - Q_n\right]$

$0 < \alpha \leq 1$

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 nth selection of action a

Choice $\alpha_n(a) = \frac{1}{n}$ results in the sample-average method

Example : Python, non_stationary

```
# Simulate non-stationarity: Drift in true rewards
for i = 1 to num_actions:
   true_rewards[i] += random.normal(0, 0.01)  # Small random drift
```

The step-size parameter α controls how much weight is given to new observations compared to past estimates. Step-Size Parameter (α) : α determines how quickly the agent updates its estimates based on new rewards.

$\alpha=0$: No learning occurs because no updates are made to the estimates.

$0<\alpha\leq1$: The agent learns from experience, with larger α values leading to faster updates and quicker adaptation to changes in the environment.

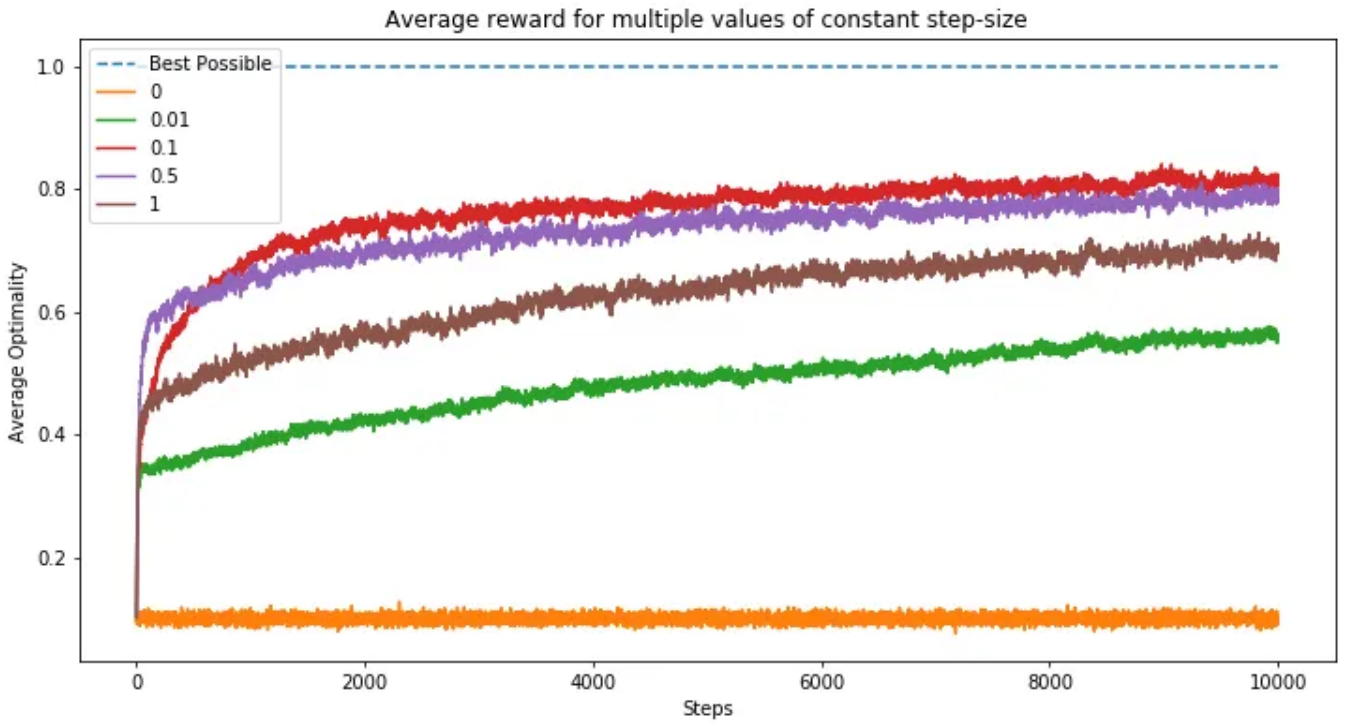
Trade-offs :

Large α (e.g., $\alpha=1$) :

+ Faster convergence to recent trends in the environment.
- Susceptible to noise or fluctuations in the environment.

Small α (e.g., $\alpha=0.01$) :

+ More stable updates, reducing sensitivity to noise.
- Slower convergence to optimal behavior.



Upper-Confidence-Bound Action Selection

Exploration is needed because there is always uncertainty about the accuracy of the action-value estimates

Action selection strategies:

- Greedy action: looks best at present - but some of the other actions may actually be better
- ε-greedy: forces the non-greedy actions to be tried - no preference for those that are nearly greedy or particularly uncertain
- Upper-Confidence-Bound Action Selection: select among the non-greedy actions *according to their potential* for actually being optimal. Take into account
  - how close their estimates are to being maximal
  - the uncertainties in those estimates

UCB

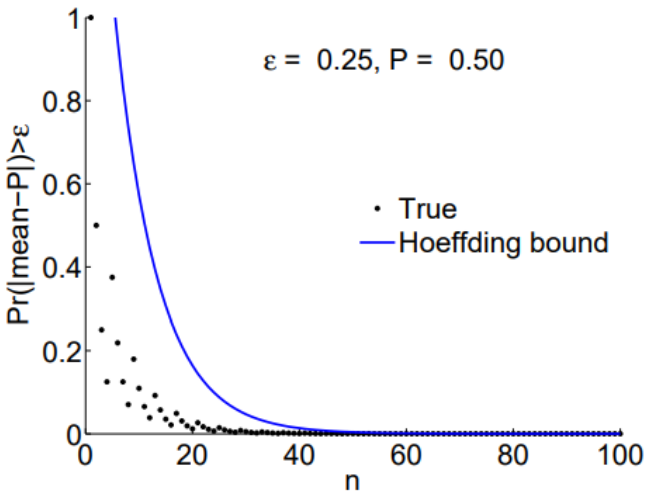
$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

$Q_t(a)$: The estimated value of action a at time t.
$N_t(a)$: The number of times action a has been selected up to time t.
t: The total number of steps taken so far.
c: A hyperparameter that controls the degree of exploration. Larger c encourages more exploration.
First Term: represents the exploitation component. Actions with higher estimated rewards are favored.
Second Term : exploration bonus. It increases for actions that have been selected fewer times ($N_t(a)$ is small), encouraging the agent to explore less-visited actions.
The term lnt ensures that the exploration bonus decreases over time but never completely disappears.

Example : Python, ucb

UCB Action Selection

Possible way: select action s.t.



Hoeffding's inequality

$$A_t = \text{argmax}_a \left(Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}} \right)$$

Exploit

Explore

t = timesteps

$N_t(a)$ = no. of times action (a) is taken

Auer, P., Cesa-Bianchi, N. & Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning 47, 235–256 (2002). https://doi.org/10.1023/A:1013689704352

Actions : a_1,a_2,a_3. Estimated rewards : $Q_t(a_1)=5$ $Q_t(a_2)=7$ $Q_t(a_3)=6$
Number of times each action has been selected: $N_t(a1)=10$ $N_t(a_2)=5$ $N_t(a_3)=8$
Total steps so far: t=23. Exploration parameter: c=2. Compute the UCB score for each action and select the best one.

Step 1: Compute the UCB Score for Each Action $UCB(a_1)=6.12$ $UCB(a_2)=8.584$
$UCB(a_3)=7.252$

$$\mathrm{UCB}(a) = Q_t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}}$$

The highest UCB score is for a_2: $A_t=a_2$

$$\mathrm{UCB}(a_1) = 5 + 2 \cdot \sqrt{\frac{\ln(23)}{10}}$$

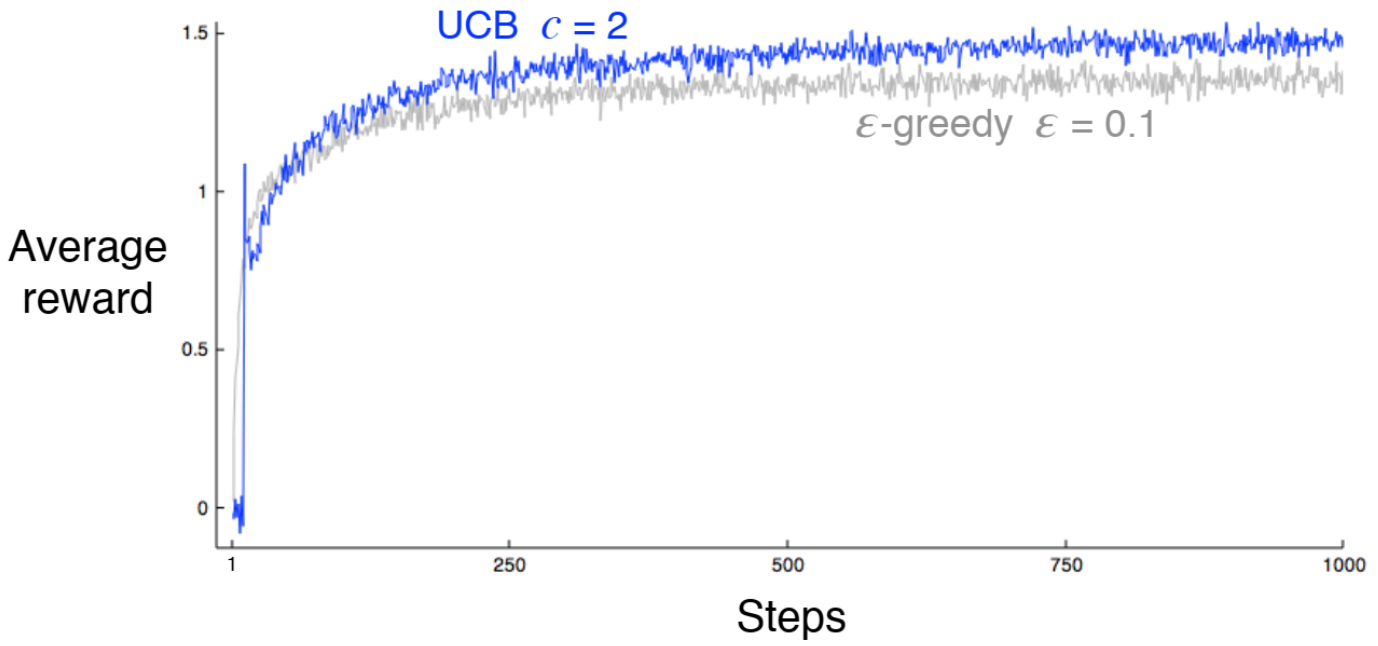
Interpretation

Action a2 was selected because it had the highest UCB score, even though its estimated reward ($Q_t(a_2)=7$) was not the highest compared to a_3 ($Q_t(a_3)=6$).

The exploration bonus for a_2 was large because it had been selected fewer times ($N_t(a_2)=5$).

Example: 10-armed bandit

Average performance of UCB



Gradient Bandit Algorithms

- ϵ-greedy or UCB - rely on action-value estimates (Q(a)),
- gradient bandit algorithms use a preference-based approach and optimize actions using gradient ascent.

The key idea is to maintain a preference function for each action, which determines the probability of selecting that action. These preferences are updated using gradient ascent to maximize the expected reward.

$$\mathbb{E}[R_t] = \sum \pi_t(a) \cdot q_*(a)$$

$$\pi_t(a) = \frac{\exp(H_t(a))}{\sum_b \exp(H_t(b))}$$

$E[R_t]$ The expected reward at time t
$q*(a)$: The true expected reward for action a.
$\pi_t(a)$: The probability of selecting action a at time t.
$H_t(a)$: The preference for action a at time t.

Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8, 229–256 (1992). https://doi.org/10.1007/BF00992696

Gradient Bandit Algorithms

Update the preferences using gradient ascent:

Where:

$$H_{t+1}(a) = H_t(a) + \alpha \cdot (R_t - \bar{R}_t) \cdot (\mathbf{1}_{A_t=a} - \pi_t(a))$$

α: The step size (learning rate).

R_t: The observed reward.

$\bar{R}_t$: The average reward up to time t, used as a baseline.

1_{At}=a: An indicator function that is 1 if At=a, and 0 otherwise.

Reward Advantage ($R_t - \bar{R}_t$) : measures how much better the observed reward R_t is compared to the baseline $\bar{R}_t$.

If $R_t > \bar{R}_t$, the preference for the chosen action A_t is increased.

If $R_t < \bar{R}_t$, the preference for the chosen action A_t is decreased.

Probability Adjustment ($1_{At=a} - \pi_t(a)$) : ensures that the update is proportional to the difference between the actual selection of the action (1At=a) and its probability of being selected (πt(a)).

Actions that are selected more frequently than their probabilities suggest receive smaller updates, while under-selected actions receive larger updates.

!!! Preferences are updated continuously based on observed rewards, allowing the algorithm to adapt to changes in the environment.

Gradient Bandit Algorithms

1. *Set up: num. of actions, learning rate, average reward.*
2. *Init: probabilities of selecting each action= 0*
3. *For each time step :*
    1. *Compute Action Probabilities : softmax function.*
    2. *Select an Action : Randomly choose action based on the computed probabilities. Actions with higher probabilities are more likely to be picked.*
    3. *Observe the Reward : come from the environment or a reward distribution associated with the action.*
    4. *Update the Baseline : calculate the running average of all rewards observed so far.*
    5. *Update Action Preferences : For the action you just took, increase its preference based on how much better (or worse) the reward was compared to the baseline. For all other actions, slightly decrease their preferences to reflect that they were not chosen.*
4. *Repeat: Continue the process for a fixed number of steps or until the algorithm converges.*
5. *Analyze Results*

Problem Setup: actions : a_1,a_2,a_3.
Init preferences: $H_0(a_1)=0$, $H_0(a_2)=0$, $H_0(a_3)=0$.
Step size: $\alpha=0.1$.
Average reward: $\bar{R}_t=5$ (initially).

$$\pi_t(a_1) = \pi_t(a_2) = \pi_t(a_3) = \frac{\exp(0)}{\exp(0)+\exp(0)+\exp(0)} = \frac{1}{3}$$

Select an Action : Sample an action from the distribution: suppose $A_t=a_2$.
Observed reward: $R_t=7$
Update Preferences :

$$H_{t+1}(a) = H_t(a) + \alpha \cdot (R_t - \bar{R}_t) \cdot (\mathbf{1}_{A_t=a} - \pi_t(a))$$

$$H_{t+1}(a_1) = 0 + 0.1 \cdot (7-5) \cdot (0 - \frac{1}{3}) = 0 - 0.1 \cdot 2 \cdot \frac{1}{3} = -0.067$$

$$H_{t+1}(a_2) = 0 + 0.1 \cdot (7-5) \cdot (1 - \frac{1}{3}) = 0 + 0.1 \cdot 2 \cdot \frac{2}{3} = 0.133$$

$$H_{t+1}(a_3) = 0 + 0.1 \cdot (7-5) \cdot (0 - \frac{1}{3}) = 0 - 0.1 \cdot 2 \cdot \frac{1}{3} = -0.067$$

Update the baseline $\bar{R}_t$ (average reward) $\bar{R}_t = \bar{R}_{t-1} + \frac{R_t - R_{t-1}}{t}$

Compute Action Probabilities: The softmax function converts preferences $H_t(a)$ into probabilities $\pi_t(a)$

$$\pi_t(a) = \frac{\exp(H_t(a))}{\sum_b \exp(H_t(b))}$$

Next …..

Example: 10-armed bandit

$\bar{R}_t \in \mathbb{R}$

$\overline{R}_t = 0$

X-axis : Number of steps (time steps). Y-axis : Percentage of optimal actions selected.

Impact of Learning Rate (α)

- Smaller α=0.1: Slower but more stable learning.
- Larger α=0.4: Faster updates but potentially less stable.

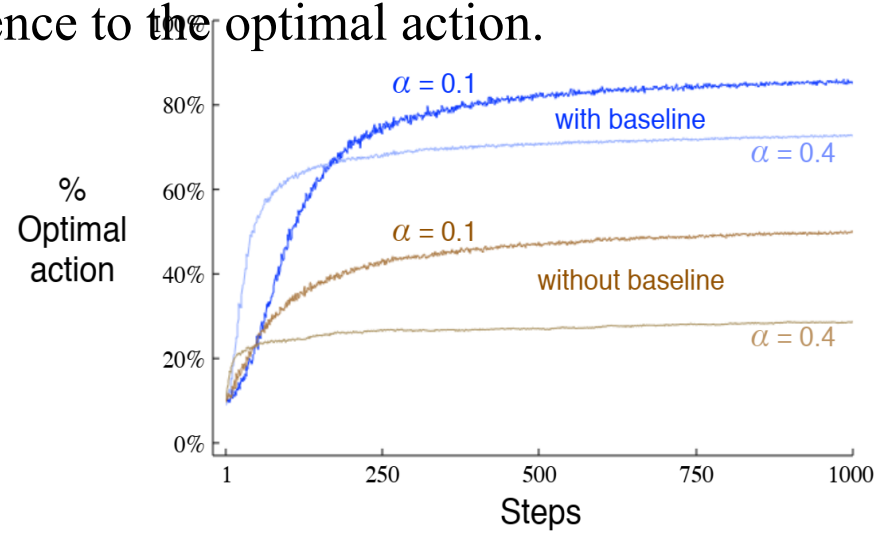
Impact of Baseline: The baseline subtracts the average reward (R¯t) from the observed reward (Rt) to reduce variance in updates.This ensures that preferences are updated based on how much better (or worse) the observed reward is compared to the average reward.

With Baseline : The blue curve shows faster convergence to the optimal action.

Using a baseline stabilizes learning, especially with a smaller step size (α=0.1).

Without Baseline : The orange curve shows slower convergence.

Without a baseline, the algorithm is more susceptible to noise in rewards.



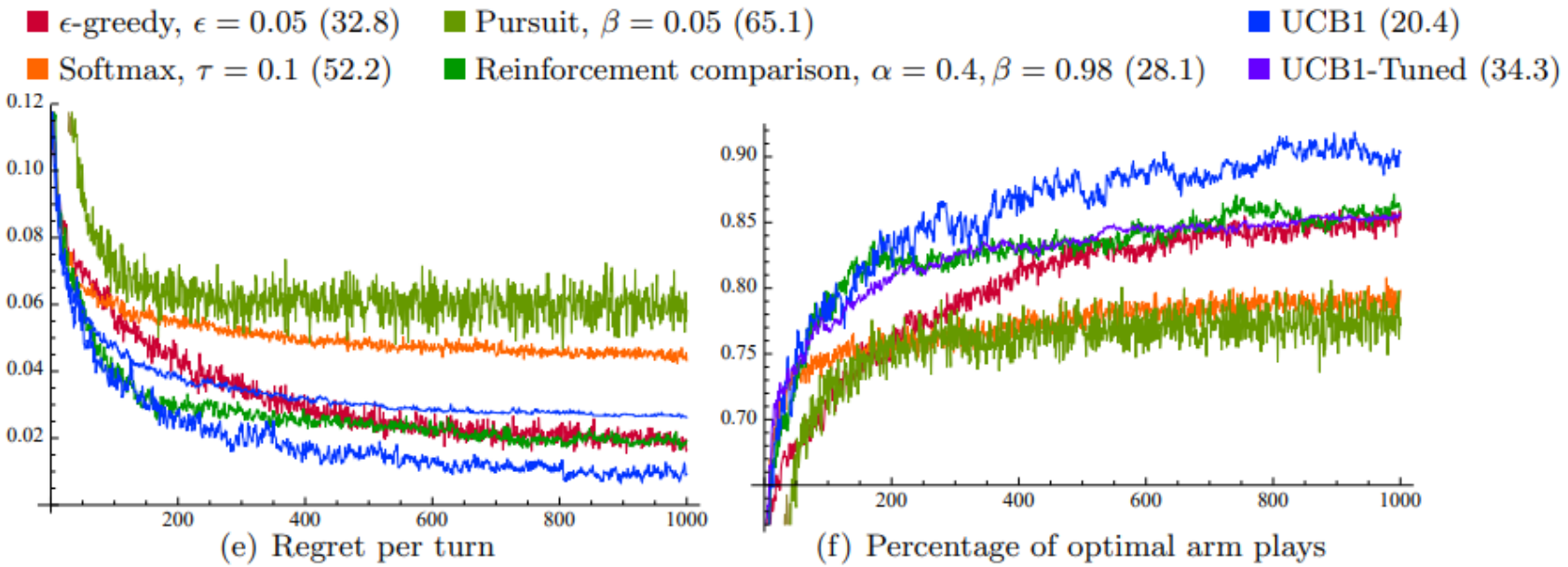
Comparison between the methods

- ε-greedy methods choose randomly a small fraction of the time
- UCB methods choose deterministically achieve exploration by subtly favoring at each step the actions that have so far received fewer samples (*according to their potential* for actually being optimal)
- Gradient bandit algorithms estimate not action values, but action preferences favor the more preferred actions in a graded, probabilistic manner using a soft-max distribution

Which of these methods is best?

Feature	Softmax	Regular Probability
Adaptability	Preferences adapt dynamically	Fixed, no adaptation
Exploration	Built-in,smooth exploration	Rigid, inefficient
Differentiability	Yes	No
Balances Exploitation/Exploration	Yes	No

Comparison between the methods



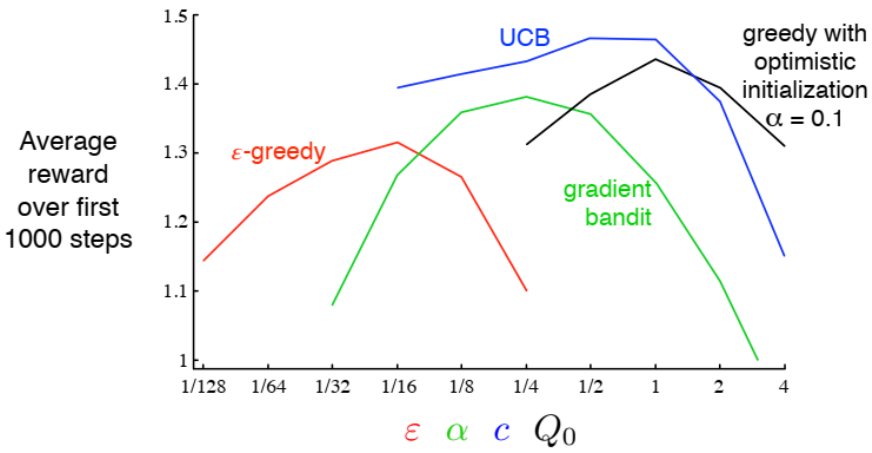
(e) Regret per turn

(f) Percentage of optimal arm plays

Kuleshov, Volodymyr, and Doina Precup. "Algorithms for multi-armed bandit problems." arXiv preprint arXiv:1402.6028 (2014).

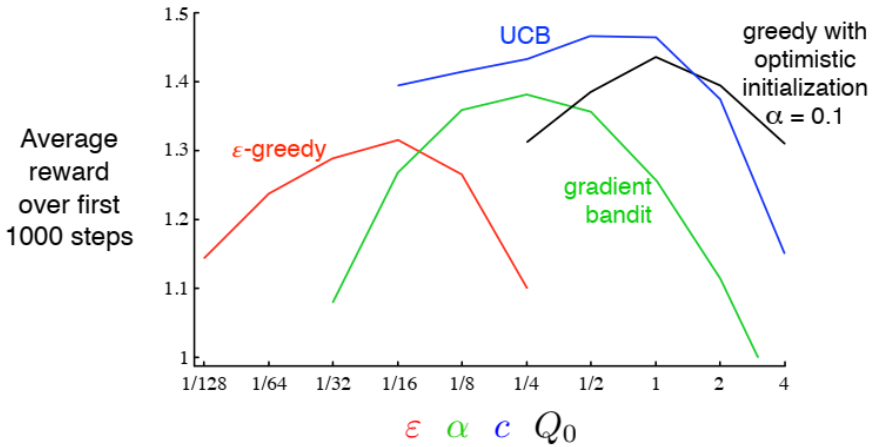
Comparison between the methods

- 10-armed bandit task
- summarize a complete learning curve by its average value over the 1000 steps
- this value is proportional to the area under the learning curve
- area under the learning curve for different bandit alg, as a function of its own parameter
  Remark: the parameter values are varied by factors of 2 and presented on a log scale

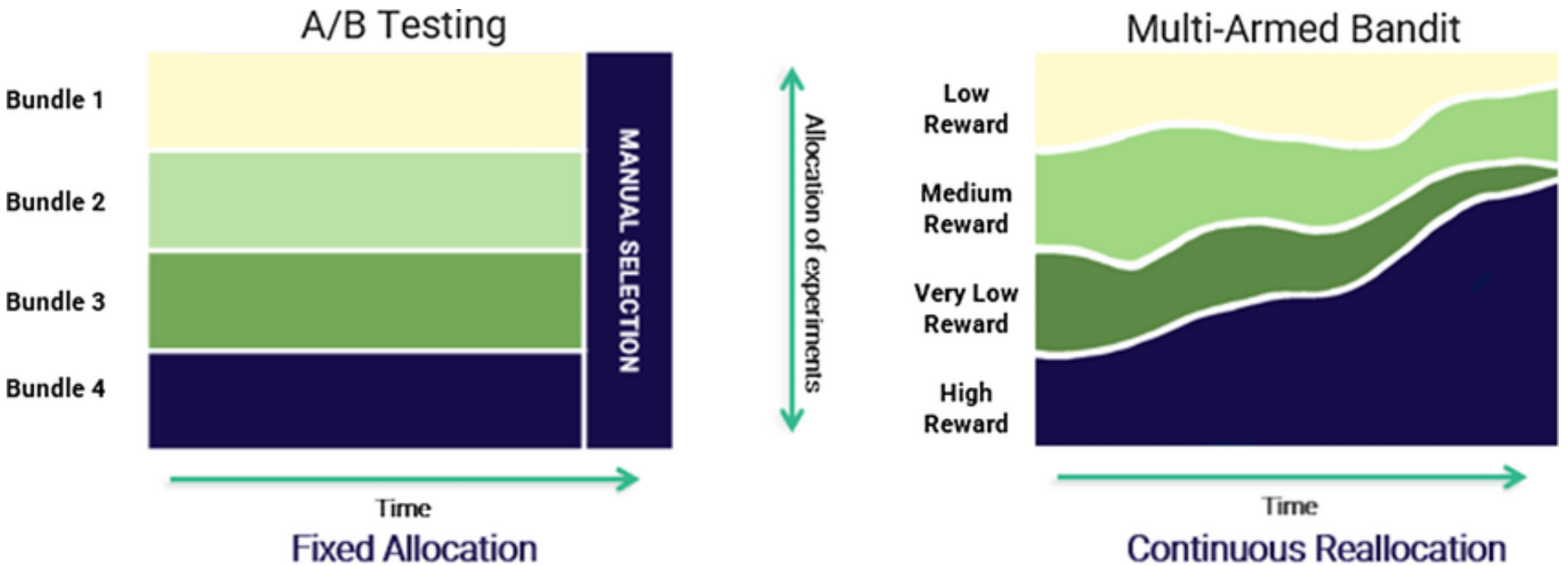


Comparison between the methods

- All the algorithms perform best at an intermediate value of their parameter
- All of these algorithms are fairly insensitive, performing well over a range of parameter values
- Overall, on this problem, UCB seems to perform best



Remark



Vinod, Balakrishna. (2022). Correction to: The age of intelligent retailing: personalized offers in travel for a segment of ONE. Journal of Revenue and Pricing Management. 21. 10.1057/s41272-021-00353-8.

Nonassociative vs Associative Tasks

- Nonassociative Tasks: there is no need to associate actions with different situations. The learner operates in a single, fixed context or situation.
  - Examples: A slot machine (multi-armed bandit problem): The learner chooses among a set of actions (e.g., pulling different arms) without considering any external "situation" or "state."
  - The goal is to find the best action (or track the best action if the environment changes over time).
- Associative Tasks: the learner must associate different actions with different situations (or states). The optimal action depends on the current situation.
  - Example: A robot navigating a maze: The best action (e.g., move left, move right) depends on the robot's current location (the state).
  - The goal is to learn a policy : a mapping from situations (states) to the best actions for those situations.

Extending Nonassociative Tasks to Associative Settings

Extend a multi-armed bandit problem to an associative setting.
Concept of states or situations .
Step 1: Introduce States: Instead of a single global context, the learner now encounters different states (or situations). Each state represents a distinct context or condition in which the learner must choose an action.
Step 2: State-Dependent Actions. The optimal action may vary depending on the current state. For example: In a multi-armed bandit problem, each arm might have different rewards depending on the state. In a navigation task, the best direction to move depends on the robot's current location.
Step 3: Learn a Policy:The learner’s goal shifts from finding a single best action to learning a policy : a mapping from states to actions. The policy specifies which action is best in each state.
Step 4: Handle Nonstationarity (Optional): the learner must also track how the best actions change over time for each state.

Why This Extension Matters

Extending nonassociative tasks to associative settings allows the learner to handle more complex environments where the best action depends on the context. This is essential for real-world applications such as:

Robotics : Choosing actions based on the robot's current location or sensor readings.

Game Playing : Selecting moves based on the current game state.

Recommendation Systems : Recommending items based on user preferences or behavior.

Example

A robot navigates a warehouse with three zones (S_1, S_2, S_3). In each zone, the robot has three possible routes (A_1, A_2, A_3) to move toward the next zone. Each route has an uncertain travel time (reward), which depends on the zone (state). The goal is to minimize total travel time

Parameters:

- States: $S = \{S_1, S_2, S_3\}$ (zones in the warehouse).
- Actions: $A = \{A_1, A_2, A_3\}$ (routes available in each zone).
- Rewards: Travel times are sampled from normal distributions:
  - S_1 : $R_{S1}(A_1)=N(5, 1^2)$, $R_{S1}(A_2)=N(7, 1^2)$, $R_{S1}(A_3) = N(6, 1^2)$
  - S_2 : $R_{S2}(A_1)= N(8, 1^2)$, $R_{S2}(A_2)=N(4, 1^2)$, $R_{S2}(A_3)= N(9, 1^2)$
  - S_3 : $R_{S3}(A_1)=N(10, 1^2)$, $R_{S3}(A_2)=N(6, 1^2)$, $R_{S3}(A_3) = N(7, 1^2)$

Goal: Learn the best action (route) for each state (zone) to minimize total travel time.

Extending the MAB Framework

1. The robot observes its current state (zone) before choosing an action.
2. Select an Action Using ε-Greedy

3. Update Q-values Based on Rewards: After selecting an action, the robot observes the reward (travel time) and updates the Q-value for the chosen action in the current state using the formula:

$$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)} \cdot (R - Q(s,a)),$$

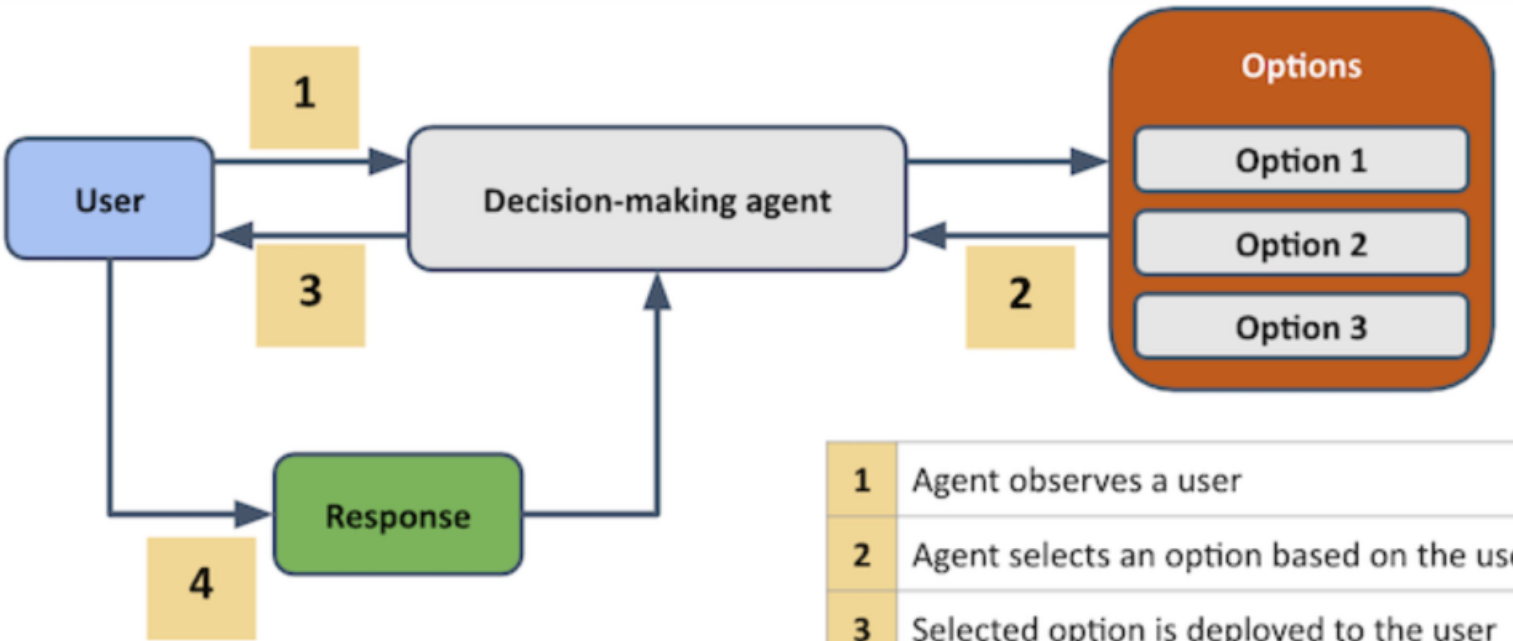
where:

- Q(s, a): Estimated travel time for action a in state s.
- N(s, a): Number of times action a has been selected in state s .
- R: Observed travel time.

4. Derive the Optimal Policy:After sufficient exploration, the robot derives the optimal policy by selecting the action with the lowest Q-value for each state.

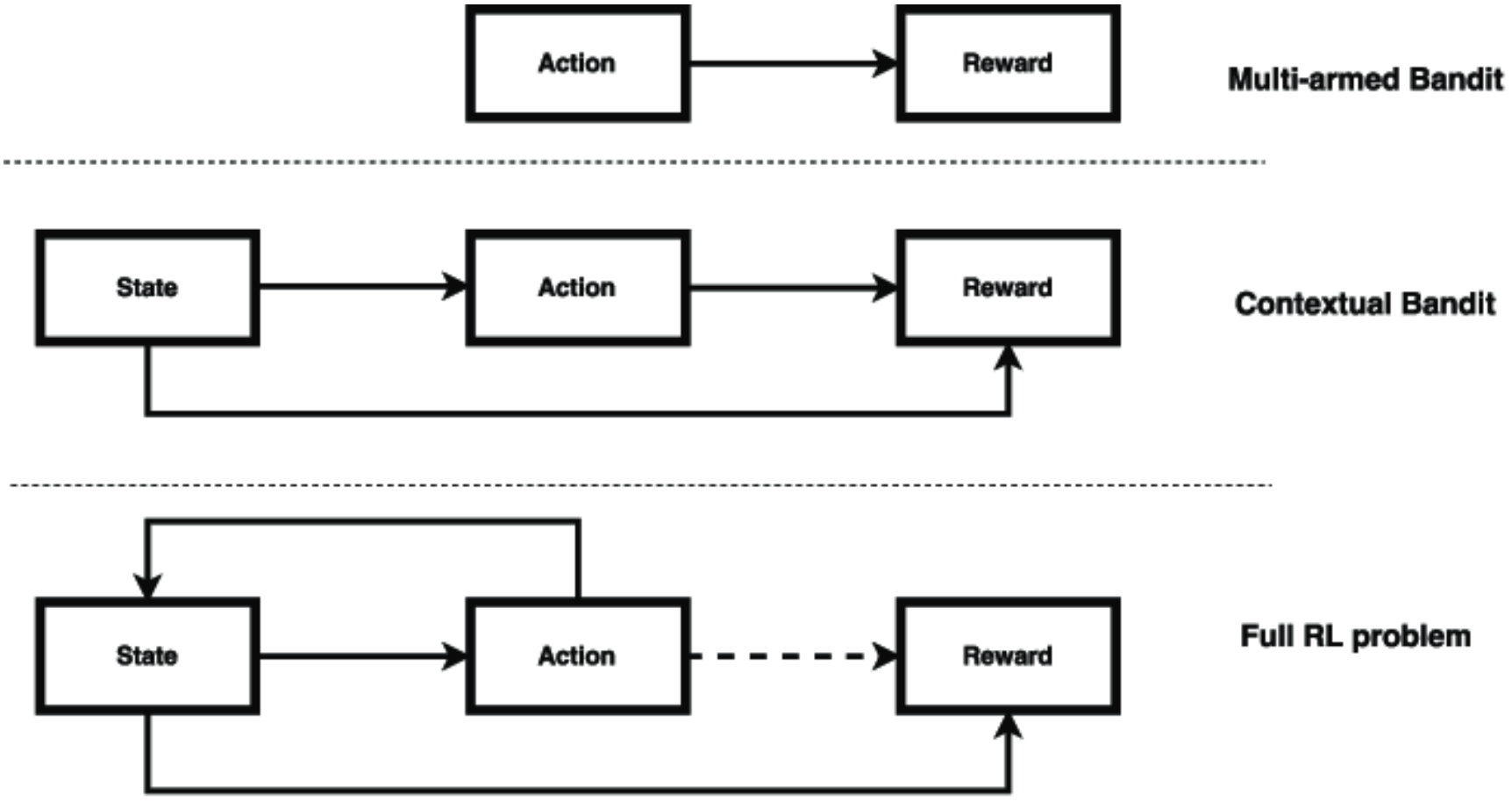
Numerical Simulation
Step 1: Initialize Q-values and Counts: $Q(S_1, A_1) = Q(S_1, A_2) = Q(S_1, A_3) = 0$,
$Q(S_2, A_1) = Q(S_2, A_2) = Q(S_2, A_3) = 0$, $Q(S_3, A_1) = Q(S_3, A_2) = Q(S_3, A_3) = 0$.
Step 2: Simulate the robot navigating through the warehouse for 10 steps, updating Q-values after each step.
Step 2.1: First Step: State: S_1. Action Selection: Use ε-greedy (epsilon = 0.1):
With probability 0.1, explore: Randomly select A_2 . Observe reward: Sample R = 7.2 from $N(7, 1^2)$.
Update Q-value: $Q(S_1, A_2) = 0 + 1/1 * (7.2 - 0) = 7.2$.
Step 2.2: Second Step. State: S_2 . Action Selection: Exploit: Select A_2 (lowest Q-value so far).
Observe reward: Sample R = 4.1 from $N(4, 1^2)$. Update Q-value: $Q(S_2, A_2) =0 + 1/1*(4.1 - 0) = 4.1$.
Step 2.3: Third Step. State: S_3. Action Selection: Explore: Randomly select A_1.
Observe reward: Sample R = 10.5 from $N(10, 1^2)$. Update Q-value: $Q(S_3, A_1) =0 + 1/1*(10.5 - 0) = 10.5$.
Step 3: Repeat for Remaining Steps: Over time, the Q-values converge to estimates of the true expected travel times.
Step 4. Learned Q-values, After 10 steps, suppose the learned Q-values are:
$Q(S_1) = [5.3, 7.2, 6.0]$, $Q(S_2) = [8.4, 4.1, 9.2]$, $Q(S_3) = [10.5, 6.1, 7.0]$.
5. Optimal Policy: The optimal policy minimizes travel time by selecting the action with the lowest Q-value for each state:
- In S_1, choose A_1 (Q-value = 5.3).
- In S_2 , choose A_2 (Q-value = 4.1).
- In S_3 , choose A_2 (Q-value = 6.1).
6. Total Travel Time: Suppose the robot follows the optimal policy for 100 steps. The average travel time per step is approximately:
Average Travel Time = (5.3 + 4.1 + 6.1)/3 = 5.17.
For 100 steps:Total Travel Time = 100 * 5.17 = 517.
Final Answer
$Q(S_1) = [5.3, 7.2, 6.0]$, $Q(S_2) = [8.4, 4.1, 9.2]$, Q(S_3) = [10.5, 6.1, 7.0],
Optimal Policy: S_1 to A_1, S_2 to A_2, S_3 to A_2}.

What is Contextual Bandits



1	Agent observes a user
2	Agent selects an option based on the user
3	Selected option is deployed to the user
4	User provides a response, which the agent learns from

https://www.linkedin.com/pulse/what-contextual-bandits-aionlinecourse-lglwc/



Steenwinckel, Bram & De Backere, Femke & Nelis, Jelle & Ongenae, Femke & De Turck, Filip. (2018). Self-Learning Algorithms for the Personalised Interaction with People with Dementia.