Zmh6339
AI & Machine Learning, CS-UH 3260
Professor Keith Ross

Ziad Hassan
January 31, 2024

Assignment 1

Part 1: Continuous Bandit Algorithm

Link to Google Colab

Here is the link to the Google Colab notebook for the code.

Pseudocode

Algorithm 1 Neural Network Algorithm for Continuous Bandit Problem

```
Initialize parameters: dimension $d$, matrix $R$, learning rate, initial samples, iterations, samples per iteration
Define $cost\_function(a, R)$
Define $generate\_random\_actions(num\_samples, d)$
Define MLPWithGrad with $input\_dim$
Define $train\_model(model, optimizer, loss\_function, actions, costs, epochs)$
Define $guided\_sampling(model, num\_samples, d, num\_candidates)$
$initial\_actions, initial\_costs \leftarrow$ Generate and calculate costs
$actions\_tensor, costs\_tensor \leftarrow$ Convert to tensors
Set up MLP model and optimizer
Train MLP model on initial samples
for each iteration do
    $new\_actions \leftarrow$ Guided sampling using MLP model
    $new\_costs \leftarrow$ Calculate costs for new actions
    Update and retrain MLP model with new data
end for
$best\_action, best\_cost \leftarrow$ Find action with minimum cost
```

Algorithm Description

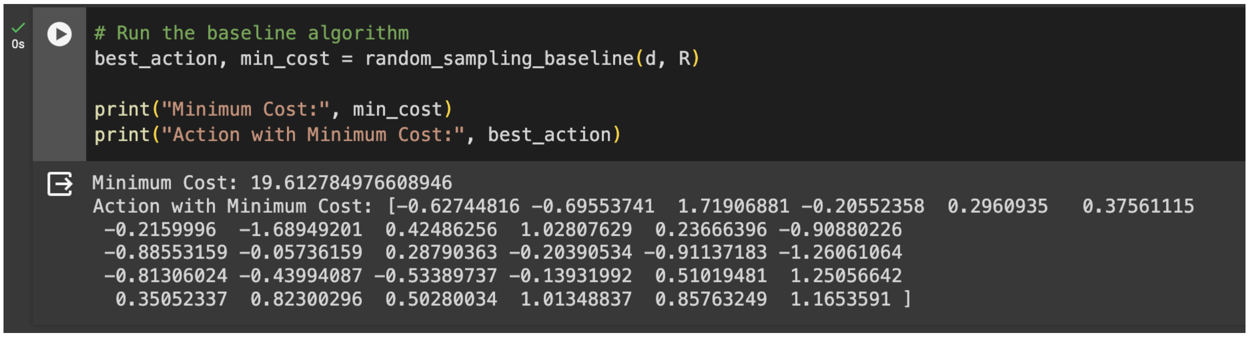
This algorithm combines initial random sampling with guided sampling by through a neural network. The process begins with randomly sampling actions within the defined action space. These initial samples provide a diverse starting point for the model without any initial bias. This phase (random sampling) would count as the "exploration."

The core of the algorithm lies in its iterative refinement process, which employs a neural network, specifically a Multi-Layer Perceptron (MLP). The MLP is trained to approximate the unknown cost function based on the initial samples. As the model learns, it guides the sampling of new actions. Here, the algorithm exploits the model's predictive power to focus on areas in the action space where lower costs are anticipated. This guided sampling, which is done by the neural network's approximation of the cost function, enables a more focused exploration of the action space. Iterating between model training and guided sampling allows the algorithm to navigate the high-dimensional space, progressively going toward the action that minimizes the cost, thereby optimizing the solution to the continuous bandit problem.

Algorithm Results

Baseline Algorithm

- $R = \mathbb{I}_{30}$

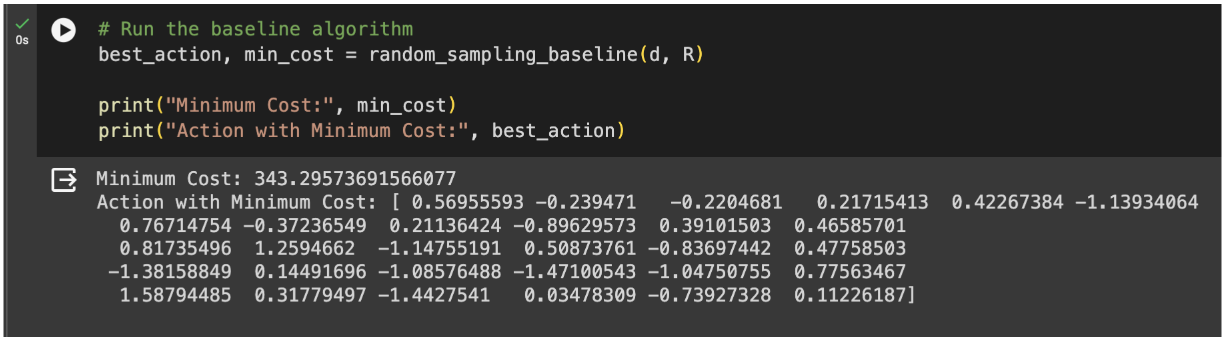


```
# Run the baseline algorithm
best_action, min_cost = random_sampling_baseline(d, R)

print("Minimum Cost:", min_cost)
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 19.612784976608946
Action with Minimum Cost: [-0.62744816 -0.69553741  1.71906881 -0.20552358  0.2960935   0.37561115
 -0.2159996  -1.68949201  0.42486256  1.02807629  0.23666396 -0.90880226
 -0.88553159 -0.05736159  0.28790363 -0.20390534 -0.91137183 -1.26061064
 -0.81306024 -0.43994087 -0.53389737 -0.13931992  0.51019481  1.25056642
  0.35052337  0.82300296  0.50280034  1.01348837  0.85763249  1.1653591 ]
```

- $R =$ random matrix



```
# Run the baseline algorithm
best_action, min_cost = random_sampling_baseline(d, R)

print("Minimum Cost:", min_cost)
print("Action with Minimum Cost:", best_action)
```

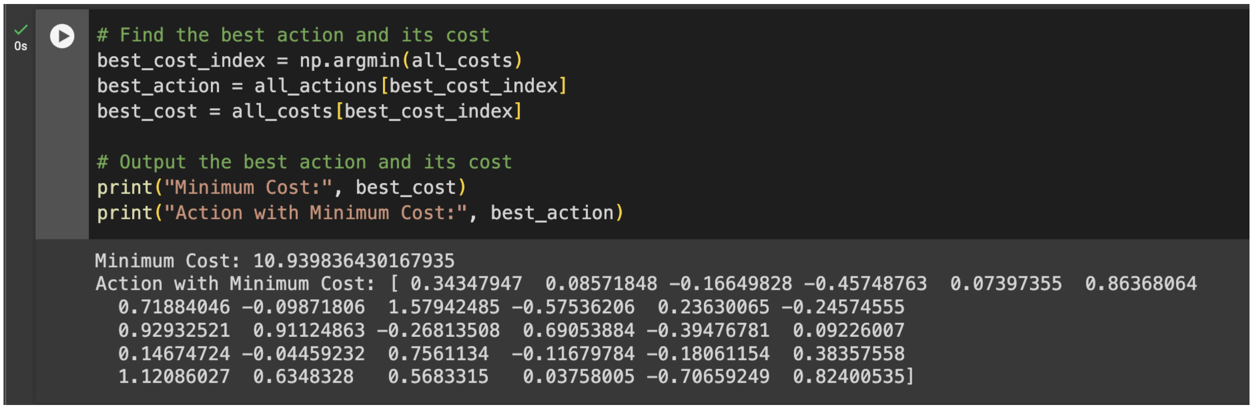
```
Minimum Cost: 343.29573691566077
Action with Minimum Cost: [ 0.56955593 -0.239471   -0.2204681   0.21715413  0.42267384 -1.13934064
  0.76714754 -0.37236549  0.21136424 -0.89629573  0.39101503  0.46585701
  0.81735496  1.2594662  -1.14755191  0.50873761 -0.83697442  0.47758503
 -1.38158849  0.14491696 -1.08576488 -1.47100543 -1.04750755  0.77563467
  1.58794485  0.31779497 -1.4427541   0.03478309 -0.73927328  0.11226187]
```

Neural Network Algorithm

Hyperparameters:

```
num_initial_samples = 100
num_iterations = 990
num_samples_per_iter = 10
learning_rate = 0.001
Total number of samples: 100 + 990*10 = 10,000
```

- $R = \mathbb{I}_{30}$

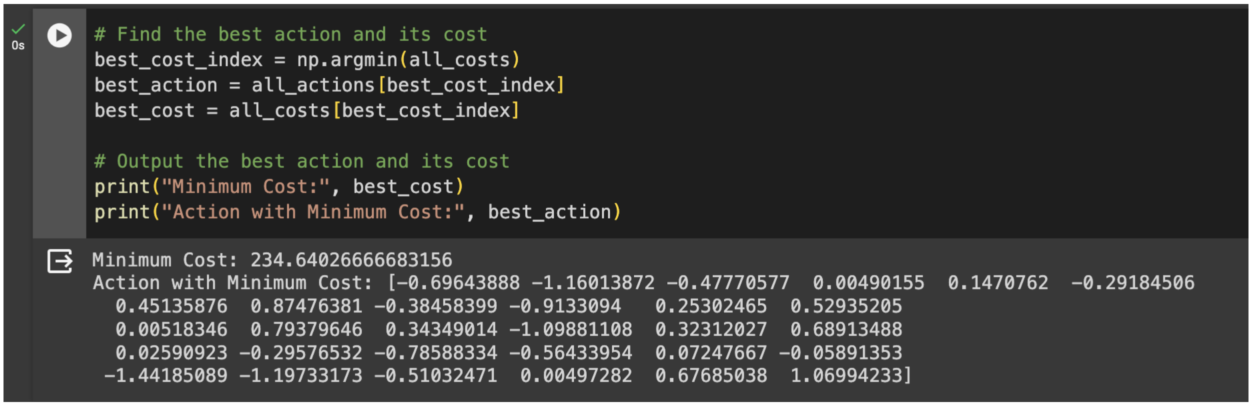


```
# Find the best action and its cost
best_cost_index = np.argmin(all_costs)
best_action = all_actions[best_cost_index]
best_cost = all_costs[best_cost_index]

# Output the best action and its cost
print("Minimum Cost:", best_cost)
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 10.939836430167935
Action with Minimum Cost: [ 0.34347947  0.08571848 -0.16649828 -0.45748763  0.07397355  0.86368064
  0.71884046 -0.09871806  1.57942485 -0.57536206  0.23630065 -0.24574555
  0.92932521  0.91124863 -0.26813508  0.69053884 -0.39476781  0.09226007
  0.14674724 -0.04459232  0.7561134  -0.11679784 -0.18061154  0.38357558
  1.12086027  0.6348328   0.5683315   0.03758005 -0.70659249  0.82400535]
```

- $R =$ random matrix



```
# Find the best action and its cost
best_cost_index = np.argmin(all_costs)
best_action = all_actions[best_cost_index]
best_cost = all_costs[best_cost_index]

# Output the best action and its cost
print("Minimum Cost:", best_cost)
print("Action with Minimum Cost:", best_action)
```

```
Minimum Cost: 234.64026666683156
Action with Minimum Cost: [-0.69643888 -1.16013872 -0.47770577  0.00490155  0.1470762  -0.29184506
  0.45135876  0.87476381 -0.38458399 -0.9133094   0.25302465  0.52935205
  0.00518346  0.79379646  0.34349014 -1.09881108  0.32312027  0.68913488
  0.02590923 -0.29576532 -0.78588334 -0.56433954  0.07247667 -0.05891353
 -1.44185089 -1.19733173 -0.51032471  0.00497282  0.67685038  1.06994233]
```

Part 2: Theory

a) *Proof.*

$$
\begin{aligned}
P(\text{greedy action}) &= 1 - P(\text{not greedy action}) \\
&= 1 - P(\text{not greedy selection } AND \text{ not greedy action}) \\
&= 1 - P(\text{not greedy selection}) \cdot P(\text{not greedy action} \mid \text{not greedy selection}) \\
&= 1 - \epsilon \cdot \left(\frac{k-1}{k}\right) \qquad {}^{*}\ \textit{Since there is only } \textbf{\textit{one}} \textit{ greedy action} \\
&= 1 - \epsilon \cdot \left(1 - \frac{1}{k}\right) \\
&= 1 - \epsilon + \frac{\epsilon}{k}
\end{aligned}
$$

■

b) i) *Proof.*

To determine the probability that the greedy action was chosen for the first time at time T, we need to consider that it was not chosen at any time before T, and that it was chosen at time T.

Thus, the following equation should be quantified:

$$P(\text{greedy at } T) = P(\text{not greedy before } T) \cdot P(\text{greedy at } T)$$

Therefore, the probability that the greedy action was chosen for the first time at time T is:

$$\begin{aligned} P(\text{greedy at } T) &= P(\text{not greedy before } T) \cdot P(\text{greedy at } T) \\ &= \left(1 - 1 + \epsilon - \frac{\epsilon}{k}\right)^{T-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \\ &= \left(\epsilon - \frac{\epsilon}{k}\right)^{T-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \end{aligned}$$

■

ii) *Proof.*

To get the expected number of steps, $\mathbb{E}[T]$, until the the greedy action is chosen for the first time is a sum over all possible time steps, each weighted by its probability of being the first time the greedy action is chosen.
Thus, the following equation should be quantified:

$$\mathbb{E}[T] = \sum_{t=1}^{\infty} t \cdot P(\text{greedy at } t)$$

Following, the equation is

$$\begin{aligned} \mathbb{E}[T] &= \sum_{t=1}^{\infty} t \cdot P(\text{greedy at } t) \\ &= \sum_{t=1}^{\infty} t \cdot \left(\epsilon - \frac{\epsilon}{k}\right)^{t-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \end{aligned}$$

It can be observed that the above equation is a geometric series, which can be simplified to the following:

$$\begin{aligned} \mathbb{E}[T] &= \sum_{t=1}^{\infty} t \cdot \left(\epsilon - \frac{\epsilon}{k}\right)^{t-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right) \\ &= \frac{1}{\left(\epsilon - \frac{\epsilon}{k}\right)^{t-1} \cdot \left(1 - \epsilon + \frac{\epsilon}{k}\right)} \end{aligned}$$

The above simplifciation is valid due to the definiton of the expected value of a geometric series.

■

c) i) *Proof.*

Firstly, denote $\max(q(1), q(2), \ldots, q(10))$ as q_*.
The algorithm will choose the greedy selection, q_* with probability $1 - \epsilon$, and a non-greedy selection with probability ϵ.
Moreover, the expected value of the an action during non-greedy selection is the average of all the action values, which is $\frac{\sum_{i=1}^{10} q(i)}{10}$.
Therefore, the long-run reward is

$$R = (1 - \epsilon) \cdot q_* + \epsilon \cdot \frac{\sum_{i=1}^{10} q(i)}{10}$$

■

ii) *Proof.*

Since $q(1), q(2), \ldots, q(10)$ are i.i.d. random variables, the expected value of a greedy action becomes $\mathbb{E}[q_*] = b$.
Moreover, since $q(a)$ is $\mathcal{N}(0,1)$, the expected average of all the action values becomes $\mathbb{E}\left[\frac{\sum_{i=1}^{10} q(i)}{10}\right] = 0$.
Therefore, the long-run reward is

$$\begin{aligned} R &= (1-\epsilon) \cdot \mathbb{E}[q_*] + \epsilon \cdot \mathbb{E}\left[\frac{\sum_{i=1}^{10} q(i)}{10}\right] \\ &= (1-\epsilon) \cdot b + \epsilon \cdot 0 \\ &= (1-\epsilon) \cdot b \end{aligned}$$

■

d)
- Restatement In any stochastic bandit problem, denoted as ν, where you have a set of actions A that is either finite or can be counted (like a list), and a fixed number of rounds or steps $n \in \mathbb{N}$, the regret R_n of following a certain strategy or policy π in this environment can be calculated as follows:

  The total regret R_n is the sum of the regrets for each individual action in A. The regret for each action a is the product of two things: the suboptimality gap Δ_a of action a (which is how much less reward you expect from action a compared to the best possible action), and the expected number of times $E[T_a(n)]$ that action a is chosen within the first n rounds.
  Mathematically, this is represented as:

  $$R_n = \sum_{a \in A} \Delta_a E[T_a(n)]$$

  This means you add up the products of the suboptimality gap and the expected number of selections for each action to find the total regret.

- Proof

  Proof.

  Before the proof, we need to redefine the above terms to be consistent with S&B's notation.
  Firstly, the set of actions, A, is the set of arms, k.
  Secondly, the suboptimality gap, Δ_a, is the difference between the expected reward of the optimal action, $\max\limits_{1 \le a \le k} q_*(a)$, and the expected reward of action $q_*(a)$:

  $$\Delta_a = \max_{1 \le a \le k} q_*(a) - q_*(a)$$

  Thirdly, the expected number of times $E[T_a(n)]$ that action a is chosen within the first n rounds is the expected number of times that action a is chosen within the first n rounds:

  $$\mathbb{E}[T_a(n)] = \mathbb{E}[N_n(a)]$$

  Since the regret is the difference in choosing the optimal action and the action chosen, it can be expressed as follows:

  $$\text{Reg}_n = n \cdot \max_{1 \le a \le k} q_*(a) - \mathbb{E}[R_n]$$

where n is the number of rounds, $\max\limits_{1\leq a\leq k} q_*(a)$ is the expected reward of the optimal action, and $\mathbb{E}[R_n]$ is the expected reward of the action chosen up to round n.
To further break down the term $\mathbb{E}[R_n]$, we can transform it into a term that descirbes actions, rather than rewards.
Thus, consider the following representation:

$$R_n = \sum_{t=1}^{n}\sum_{a=1}^{k} R_t(a)\cdot \mathbb{1}_{A_t=a}$$

Now, the regret can be expressed as follows:

$$\begin{aligned}\text{Reg}_n &= n\cdot \max_{1\leq a\leq k} q_*(a) - \mathbb{E}[R_n]\\ &= n\cdot \max_{1\leq a\leq k} q_*(a) - \mathbb{E}\left[\sum_{t=1}^{n}\sum_{a=1}^{k} R_t(a)\cdot \mathbb{1}_{A_t=a}\right]\end{aligned}$$

Using properties of the expected value and that we are summing over n time steps, the above equation can be simplified to the following:

$$\begin{aligned}\text{Reg}_n &= n\cdot \max_{1\leq a\leq k} q_*(a) - \mathbb{E}\left[\sum_{t=1}^{n}\sum_{a=1}^{k} R_t(a)\cdot \mathbb{1}_{A_t=a}\right]\\ &= \sum_{a=1}^{k}\sum_{t=1}^{n}\mathbb{E}\left[\max_{1\leq a\leq k} q_*(a) - R_t(a)\cdot \mathbb{1}_{A_t=a}\right]\end{aligned}$$

The expected reward on round t is conditional on the action chosen on round t, A_t.
Thus, the expected value term can be expressed as follows:

$$\begin{aligned}\mathbb{E}\left[\max_{1\leq a\leq k} q_*(a) - R_t(a)\cdot \mathbb{1}_{A_t=a}|A_t\right] &= \mathbb{1}_{A_t=a}\mathbb{E}\left[\max_{1\leq a\leq k} q_*(a) - R_t(a)|A_t\right]\\ &= \mathbb{1}_{A_t=a}\left(\max_{1\leq a\leq k} q_*(a) - q(a)\right)\\ &= \mathbb{1}_{A_t=a}\cdot \Delta_a\end{aligned}$$

The term $\sum_{t=1}^{n}\mathbb{1}_{A_t=a}$ is the number of times that action a is chosen within the first n rounds, $N_n(a)$.
Therefore, we get the final result that the formula for the regret is

$$\text{Reg}_n = \sum_{a=1}^{k}\Delta_a\cdot \mathbb{E}[N_n(a)]$$

■

e)
- Algorithm Restatement in S&B Terms
  (a) **Initialization:**
    - For each action a (in $1, 2, ..., k$):
      - Initialize action-value estimate $Q_t(a) = 0$.
      - Initialize the number of times action a has been chosen, $N_t(a) = 0$.

  (b) **Loop for each step $t = 1, 2, 3, ...$:**
    - Select action A_t using the UCB criterion:

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

      where c is a confidence level parameter.

- Take action A_t, observe reward R_t.
- Update the action-value estimate for A_t:

$$Q_{t+1}(A_t) = Q_t(A_t) + \frac{1}{N_t(A_t)}(R_t - Q_t(A_t))$$

- Increment $N_t(A_t)$.

- Proof Restatement

In a k-armed bandit problem using the UCB algorithm, for any time step t, the expected regret R_t is bounded by:

$$R_t \leq 3\sum_{i=1}^{k} \Delta_i + \sum_{i:\Delta_i>0} \frac{16\log(t)}{\Delta_i}$$

where $\Delta_i = q_*(a^*) - q_*(a_i)$ is the regret associated with choosing action a_i instead of the optimal action a^*.

(a) **Optimal Action Assumption:**
Assume that the first arm a_1 is optimal, so $q_*(a_1) = q_*(a^*)$, where $q_*(a)$ represents the true value of action a.

(b) **Regret Decomposition:**
The total regret after t steps is:

$$R_t = \sum_{i=1}^{k} \Delta_i E[N_t(a_i)]$$

where $E[N_t(a_i)]$ is the expected number of selections of action a_i up to time t.

(c) **Defining "Good" Event G_i:**
Define the good event G_i for a suboptimal arm a_i as:

$$G_i = \left\{ q_*(a_1) < \min_{\tau \leq t} UCB(a_1, \tau) \right\} \cap \left\{ \hat{q}_\tau(a_i) + \sqrt{\frac{2\log(1/\delta)}{N_\tau(a_i)}} < q_*(a_1)\, \forall\, \tau \leq t \right\}$$

where $\hat{q}_\tau(a_i)$ is the estimated value of action a_i at time τ, $N_\tau(a_i)$ is the number of times action a_i has been selected up to time τ, and δ is a confidence level parameter.

(d) **Bounding $E[N_t(a_i)]$ under G_i:**
Show that under G_i, a_i is selected at most u_i times, where u_i is a function of Δ_i and δ. This leads to:

$$E[N_t(a_i)|G_i] \leq u_i$$

(e) **Bounding Probability of G_i^c:**
The probability of the complement event G_i^c (where the good event does not occur) is small. It is bounded using concentration inequalities.

(f) **Expected Selection Bound:**
Combine these results to express $E[N_t(a_i)]$:

$$E[N_t(a_i)] = E[N_t(a_i)|G_i]P(G_i) + E[N_t(a_i)|G_i^c]P(G_i^c) \leq u_i + tP(G_i^c)$$

(g) **Choosing u_i and c:**
The choice of u_i is made to ensure that the upper confidence bound for suboptimal arm a_i is properly controlled. A natural choice for u_i, considering the balance between exploration and exploitation, is given by:

$$u_i = \left\lceil \frac{2\log(1/\delta)}{(1-c)^2\Delta_i^2} \right\rceil$$

where δ is the confidence level parameter, and c is a constant.
In the proof, c is chosen to be $1/2$ somewhat arbitrarily but in a way that balances the two terms in the regret bound. This choice leads to a simplification of the regret bound while maintaining a balance between exploration and exploitation.

(h) **Finalizing the Bound on R_t:**
With u_i and c chosen, the final bound on R_t is derived by substituting these values into the regret decomposition:

$$R_t \leq \sum_{i:\Delta_i>0} \Delta_i \left(\left\lceil \frac{2\log(1/\delta)}{(1-c)^2\Delta_i^2} \right\rceil + tP(G_i^c) \right)$$

Applying the choice of $c = 1/2$, the final bound simplifies to:

$$R_t \leq 3\sum_{i=1}^{k} \Delta_i + \sum_{i:\Delta_i>0} \frac{16\log(t)}{\Delta_i}$$

f) We consider a stochastic k-armed bandit problem where the goal is to maximize cumulative rewards over time.
We use the Upper Confidence Bound (UCB) algorithm for arm selection. Let Q_n be the average reward at time n for a chosen arm, and let a^* be the optimal arm.
We aim to prove that:

$$\lim_{n\to\infty} \mathbb{E}[Q_n] = q_*(a^*)$$

where $q_*(a^*)$ is the expected reward of the optimal arm.

Proof.

- Theorem 7.1 Recap
  Theorem 7.1 provides an upper bound on the regret R_n for the UCB algorithm:

  $$R_n \leq 3\sum_{i=1}^{k} \Delta_i + \sum_{i:\Delta_i>0} \frac{16\log(n)}{\Delta_i}$$

  where Δ_i is the difference between the expected reward of the optimal arm and arm i.
- Definition of Q_n
  Q_n is defined as the average of rewards received from taking action a up to time t:

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

  where R_i is the reward received at time i, A_i is the action taken at time i, and $\mathbb{1}_{A_i=a}$ is the indicator function.

By the law of large numbers, for each arm a, the average reward converges to the expected reward $q_*(a)$ as n increases.

Therefore,

$$\frac{\sum_{i=1}^{n} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{n} \mathbb{1}_{A_i=a}} \to q_*(a)$$

as $n \to \infty$.

The UCB algorithm selects arms based on the estimated reward and the uncertainty in that estimate. Over time, it favors arms with higher estimated rewards.

For the optimal arm a^*, as $n \to \infty$, the UCB algorithm will increasingly favor this arm, assuming it correctly identifies it. Therefore, the frequency of selecting arm a^*, denoted by $N_{a^*}(n)$, will dominate over other arms, leading to:

$$\frac{N_{a^*}(n)}{n} \to 1$$

and for each suboptimal arm a,

$$\frac{N_a(n)}{n} \to 0$$

Given the dominance of the optimal arm in selections, the limit of the expected average reward $\mathbb{E}[Q_n]$ converges to the expected reward of the optimal arm:

$$\lim_{n \to \infty} \mathbb{E}[Q_n] = q_*(a^*)$$

Under the UCB algorithm, the expected average reward for the arm selected by the algorithm converges to the expected reward of the optimal arm as the number of trials n goes to infinity.

■