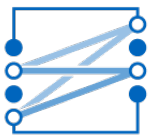
DEPARTMENT OF COMMUNICATION ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Research Internship in Communication Engineering

Algorithm and Regret bounds for Multi-armed Bandit

Lewis Chua Kah Leong

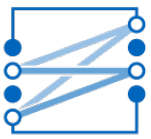


DEPARTMENT OF COMMUNICATION ENGINEERING

TECHNISCHE UNIVERSITÄT MÜNCHEN

Research Internship in Communication Engineering

Author:	Lewis Chua Kah Leong
Supervisor:	Abdalla Ibrahim
Submission Date:	Submission date



Abstract

Multi-Armed Bandit is a machine learning procedure in which an agent must select actions (arms) to maximize its cumulative reward in the long run. In each round, the agent receives information about the current state (context) and then selects an action based on this information and the experience gained in the previous rounds. After each round, the agent will be able to receive a particular reward that is associated with the chosen action. The MAB can be split into two major types, Stochastic Bandit and Bayesian Bandit. Both bandits will be discussed, and respective research and experiment will be conducted on the different variations of each Bandit. A regret analysis will also be conducted for one of the mentioned bandits. A comparison was also conducted on how different distributions, such as Laplace, Bootstrap, and Langevin, will also affect the performance of a Bayesian bandit.

Contents

1 Introduction

Multi-Armed bandit (MAB) is a problem often discussed in probability theory and machine learning[1]. It is a problem with constant, limited resources between competing (alternative) decisions. The problem is that the set exists when the characteristics of each option are only partially known at the time of allocation. They are created to maximize the expected return and can be better understood over time or by allocating resources to options. This is a classic reinforcement learning problem that illustrates the trade-offs between exploration and exploitation. The term "exploration and exploitation" comes from the context of slot machines, where a player must decide which machine to play, how many times to play each machine, and in what order to play the current machines or whether to continue with his machine or try another machine.

1.1 Multi Armed Problem

In such a MAB problem, the algorithm will provide a random reward from a specific probability distribution that is not known. This probability distribution is known as the prior distribution. The main objective of the algorithm is to maximize the sum of rewards and regret that is generated through a series of learning decisions.
An classic example of a MAB problem is a slot machine. Given a slot machine with a random distribution of rewards, a player's objective will be to try to maximize his earnings(reward) through a series of lever pulls. The crucial decision faced by the player at each attempt is to maximize the **Exploitation** of the machine that gives the highest expected payoff and **Exploration** to obtain more information about the expected payout of the other machines. This trade-off between exploration and exploitation is often faced in machine learning. In practice, MAB is often used to model problems such as clinical trials, anomaly detection[2], and even big companies like Netflix also use MAB to generate recommendations[3] to their viewers and even clinical trials[4, pp. 2–4]. Two simple MAB problem example can be shown as follows;

In such a MAB problem, the algorithm will provide a random reward from a specific probability distribution that is not known. This probability distribution is called the priority distribution. The main goal of the algorithm is to maximize the sum of rewards and regrets generated by a series of learning decisions. A slot machine is a classic example of a MAB problem. In a slot machine with a random distribution of rewards, the player aims to maximize his winnings (reward) through a series of lever turns to maximize through a series of lever turns. The critical decision the player must make on each attempt is to maximize the exploitation of the machine that offers the highest expected payout and to Exploration

to obtain more information about the expected payoff of the other machines. This trade-off between Exploration and exploitation is often made in machine learning. In practice, MAB is commonly used to model problems such as clinical trials and [2], and even big companies like Netflix also use MAB to generate recommendations[3] to their viewers and even clinical trials[4, pp. 2–4]. Two simple MAB problem examples can be illustrated as follows;

Given a scenario of a news website, when a new user arrives, the website will pick an article header to show and observe whether the user clicks on this header. The site's goal is to maximize the total number of clicks under the assumption that it has to learn the user's behavior which follows a statistical pattern it does not know.

Another example would be the usage of a dynamic pricing system. Given that a store is selling digital goods online when a new customer arrives, the store will have to decide on a price to be offered to the customer. The customer will have the choice to either buy or leave forever. Similarly to the previous example, the goal will be to maximize profit while learning the customer's shopping trends and habits.

Example	Action	Reward
News Website	An article to display	1 if clicked, 0 otherwise
Dynamic Pricing	A price to offer	p if sale, 0 otherwise

Table 1.1: Action and Reward for the examples on news website and dynamic pricing

Table 1.1 shows the table for the relationship between the example and the subsequently action and reward that will be selected. For the example of News Website, the action will be determining which article to display to the user and the reward will be a Bernoulli reward of 1 if it is clicked and 0 if otherwise. Similarly, for the Dynamic pricing example, the action will be deciding what price to offer and the reward will be the sale price, p. In terms of MAB, the bandit would be the overall class encapsulating the action choice and reward generation. While the action and reward will be a function indicated inside the class bandit.

1.2 Reinforcement Learning

As mentioned previously, MAB is a reinforcement learning algorithm. In a reinforcement learning method, the algorithm will attempt to discover which action yields the best reward and how will this reward be affected by the next following choice of action. One of the key goals of reinforcement learning would be to find the balance between Exploration and Exploitation in order to obtain the best possible reward. Another goal would be that the algorithm could explicitly consider the whole given problem with a specific goal to work towards while training in a completely new and foreign environment. There are many real-life examples that are mentioned in the book by Richard S. Sutton and Andrew G. Barto [5] on "Reinforcement Learning: An introduction". One such example that was mentioned in the book would be the choice that would be made by a chess player. The choice would be determined both by planning and anticipating what the opponent would do and the possible

counteraction(exploration) by learning from the opponent existing's moves(exploitation).

1.3 Reward

In reinforcement learning, rewards are a vital part of the learning process. Rewards signal to a bandit algorithm what valuable outcome it has obtained, indicating which ones should be exploited when the same condition is visited. Rewards play a critical role in reinforcement learning by motivating the bandit to learn and explore. With rewards, a bandit algorithm would have a way of knowing which results are significant and would be likely to make progress in achieving the most optimal results. Rewards provide feedback to the bandit algorithm, allowing it to learn from its algorithm and adapt its reward generated over time. There are two main types of rewards in reinforcement learning: deterministic and stochastic. Deterministic rewards always occur when a bandit algorithm reaches a particular condition and a specific rule is taken. Mathematically, if the bandit algorithm is in an exploration state and receives a reward, it will always receive the same reward if it is the most optimal by exploiting it. On the other hand, stochastic rewards may or may not occur when a condition is reached or a specific control is taken. For example, the reward function is stochastic if the running robot has a 50% chance of receiving a point every time it completes a lap around the track. Stochastic rewards are a more general formulation than deterministic rewards; in games with stochastic rewards, generally, more exploration is required of the agent, and it takes longer for an agent to learn the optimal approach.
The reward that is generated is either a Bernoulli or Normal distribution reward depending on the type of bandit that is involved. And it is worth noting that both Bernoulli and Normal random values also belong to a wider class of random variable which is called, ***Sub-Gaussian***. A Sub-Gaussian can be defined as follows, A random variable X is a σ-subgaussian if and for all $\lambda \in \mathcal{R}$, it holds that,

$$\mathbb{E}\left[\exp \lambda X\right] \leq \exp \frac{\lambda^2 \sigma^2}{2}$$

With the two following statement holding true;
1. If X is Gaussian with mean zero and variance, σ^2, then X is a σ-subgaussian.
2. If X has mean zero and $X \in [a, b]$ then X is $\frac{b-a}{2}$ subgaussian.

1.4 Regret

The benchmark of determining a particular method's quality is by looking at the regret. Regret is determined by the action that should have been taken but failed to do so. Therefore, the value of regret can be understood as the proportion between the probability of not taking an action that is deemed optimal and the probability of taking an optimal action. From the point of view of mathematics, regret can be expressed as the difference between the reward from an action and the reward from an optimal action. A regret of 0 is deemed the best, as the lower the regret, the better the reward and learning outcome of the bandit. Therefore, it

is crucial to study the minimization of regret, as shown below. With μ^* and μ_i given by the optimal/best mean reward and reward at the current instance, respectively.

$$\begin{aligned} R_t &= t\mu^* - \mathbb{E}\left[S_t\right] \\ &= t\mu^* - \mathbb{E}\left[\sum_{t=1}^{t} X_t\right] \\ &= \sum_{a\in\mathbb{A}}\sum_{t=1}^{T}\mathbb{E}[(\mu^* - X_t)\mathbb{I}\{A_t = a\}] \end{aligned} \tag{1.1}$$

As shown in 1.1, the regret can be obtain as the expected value of the difference between the optimal reward and the current reward which will be represented by the indicator function to know the exact arm that causes the current action.
The expected reward in round t conditioned on A_t is μ_{A_t} which can be shown as follows;

$$\begin{aligned} \mathbb{E}[(\mu^* - X_t)\mathbb{I}\{A_t = a\}|A_t] =& \mathbb{I}\{A_t = a\}\mathbb{E}[\mu^* - X_t|A_t] \\ =& \mathbb{I}\{A_t = a\}(\mu^* - \mu_{A_t}) \\ =& \mathbb{I}\{A_t = a\}(\mu^* - \mu_a) \\ =& \mathbb{I}\{A_t = a\}(\Delta_a) \end{aligned} \tag{1.2}$$

By substituting back into Equation 1.1, the regret formula can be simplified into,

$$R_n = \sum_{i=1}^{n}\Delta_i\mathbb{E}[T_i] \tag{1.3}$$

Where $\Delta_i = \mu^* - \mu_i$ is denoted as the sub-optimality gap between the mean of action i and the optimal action. X_{A_t} is denoted as the reward of the action at instance t. The indicator function, $\mathbb{I}$ has the following properties;

$$\mathbb{I}\{A_t = i\} = \begin{cases} 1, & \text{if } A_t = i \\ 0, & \text{otherwise} \end{cases} \tag{1.4}$$

T_i indicates the number of times action, i has been played after round, t.

$$T_i(t) = \sum_{s=1}^{t}\mathbb{I}\{A_s = i\} \tag{1.5}$$

Two lemma will also be discussed and will be utilised for the regret decomposition that will be conducted in the next chapter.
Lemma 1. For any bandit, π and stochastic environment, $\mathcal{V}$ with $\mathcal{A}$-finite or countable and horizon, $n \in \mathbb{N}$, the regret R_n of bandit π in $\mathcal{V}$ satisfies,

$$\begin{aligned} R_n &= \max_{i=1,\ldots,n}\mu_i^* T - \sum_{i=1}^{n}\mu_i\mathbb{E}[T_i] \\ &= \sum_{a\in\mathcal{A}}\Delta_a\mathbb{E}[T_a] \quad \text{(Regret decompsition)} \end{aligned}$$

Lemma 2. If X is σ-subgaussian, then for any $\epsilon \geq 0$

$$\mathbb{P}(X \geq \epsilon) \leq \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$$

2 Multi-Armed bandit Algorithm

The goal of a MAB algorithm is to minimize the regret as shown in Equation 1.3 while maximizing the reward. Low regret will result in a higher probability of reward generated.

2.1 Stochastic Bandit

In a stochastic bandit problem, each bandit arm, $k \, \epsilon \, [n]$ is associated with a random normal distribution $\nu_i \, \epsilon \, [0, 1]$ and the reward generated by each bandit arm k are assumed to be i.i.d from ν_i. The goal would be to maximize its total reward over T rounds. Observation is only made for the reward for the selected action based on the current arm. Other rewards that are made for other actions will not be observed. As such the cumulative regret can be extended from Equation 1.3 and be defined as such $n \in \mathbb{N}$;

$$R_t = t\mu^* - \mathbb{E}[\sum_{t=1}^{t} \mu_t] \tag{2.1}$$

μ^* denotes the optimal/best reward of all arms until time, t. And μ_t is the reward by the chosen arm at the time, t. The optimism principle is that by using a concentration argument so that each true mean, $\mu(a)$ can be in between the interval of $\hat{\mu}(a) \pm conf(a)$.
Therefore, the regret, R_n, is random as it depends on the randomness in reward generated by the arms. As such, the value of $\mathbb{E}[R_n]$ will be random as well. Since the rewards generated are stochastic as well, it would be natural to estimate the mean.
There are many types of stochastic bandits, such as ϵ-greedy bandit (subsection 2.1.2), uniform bandit (subsection 2.1.1), gradient ascent bandit (subsection 2.1.6), and Upper Confidence Bound (UCB) bandit (subsection 2.1.4). The different types of bandits will be further explored and discussed in the following sections.

Algorithm 1: Stochastic Bandit

Parameters: K arms, T rounds, i.i.d reward distribution ν_i for each arm, i.
for *each round*, $t \in [T]$ **do**
1. *Exploration*: The algorithm picks an action α_t based on the current arm, i.
2. A reward, $X_i \in [0, 1]$, based on the action α_i is sampled independently from ν_i.
3. *Exploitation*: The algorithm obtains the reward, R_i, and observes nothing else.

end

2.1.1 Explore then Commit Bandit

In Explore then Commit(ETC) bandit, the arms will be explored uniformly at the same rate regardless of the observation that was made previously. The best optimal arm will be picked for exploitation. ETC is characterized by the number of times it explores each arm, denoted by a natural number m. Because there are k actions, the algorithm will explore for mk rounds before choosing a single action for the remaining rounds. This is to ensure that the regret will be minimized. One major problem faced with the ETC algorithm is that the arms have huge differences, which will significantly impact the performance during the Exploration phase resulting in a poorly optimized regret. However, this is a straightforward bandit implementation and is often not used for a MAB problem, as a much better algorithm can be used instead.

Algorithm 2: Explore then Commit Bandit

Input m.

In round t, choose action

$$A_t = \begin{cases} \text{(t mod k)+1}, & \text{if } t \leq mk; \\ argmax_i \hat{\mu}_i(mk), & \text{t > mk} \end{cases}$$

The choice of **m** is crucial to determine how long will the bandit explore before proceeding to exploit the result. Having a huge **m** will allows the bandit to explore for a longer period. A regret analysis can be performed for ETC given a average reward received from arm, i after round, t as $\hat{\mu}_i(t)$.

$$\hat{\mu}_i(t) = \frac{1}{T_i(t)} \sum_{s=1}^{t} \mathbb{I}\{A_s = i\} X_s$$

When ETC algorithm is interacting with a $1 - subgaussian$ bandit and $1 \leq m \leq \frac{t}{k}$. With t as the time horizon and k being the total number of arms.
A regret can be bounded as follows;

$$R_t \leq m \sum_{i=1}^{k} \Delta_i + (t - mk) \sum_{i=1}^{k} \Delta_i \, exp\left(-\frac{m\Delta_i^2}{4}\right) \tag{2.2}$$

The proof for Equation 2.2 can be obtained by assuming a bandit without loss of generality that the first arm is optimal, such that $\mu^* = \mu_1 = \underset{i}{max}\mu_i$. By substituting the result of Equation 1.3 into Equation 2.2, the following equation can be obtained.

$$\sum_{i=1}^{k} \Delta_i \mathbb{E}[T_i] \leq m \sum_{i=1}^{k} \Delta_i + (t - mk) \sum_{i=1}^{k} \Delta_i \, exp\left(-\frac{m\Delta_i^2}{4}\right) \tag{2.3}$$

In the first mk rounds,the bandit select each actions exactly m times. Subsequently it will select a single action maximising the average reward during exploration as shown in algorithm 2. Thus,

$$\begin{aligned} \mathbb{E}[T_i(t)] &= m + (t - mk)\mathbb{P}[A_{mk+1} = i] \\ &\leq m + (t - mk)\mathbb{P}[\hat{\mu}_i(mk) \geq \underset{j \neq i}{max}\hat{\mu}_j(mk)] \end{aligned} \tag{2.4}$$

A bound can be obtained from the inequality on the right hand side of Equation 2.4 with the initial assumption that the first arm is optimal.

$$\mathbb{P}[\hat{\mu_i}(mk) \geq \max_{j \neq i} \hat{\mu_j}(mk)] \leq \mathbb{P}[\hat{\mu_i}(mk) \geq \hat{\mu_1}(mk)] \tag{2.5}$$

By multiplying the right hand side of Equation 2.5 by Δ_i, one can obtain the following equation,

$$\hat{\mu_i}(mk) \cdot \Delta_i \geq \hat{\mu_1}(mk) \cdot \Delta_i \tag{2.6}$$

From Equation 2.6, the bound can be equates to the following equation,

$$\begin{aligned} \mathbb{P}[\hat{\mu_i}(mk) \geq \max_{j \neq i} \hat{\mu_j}(mk)] &\leq \mathbb{P}[\hat{\mu_i}(mk) \geq \hat{\mu_1}(mk)] \\ &= \mathbb{P}[\hat{\mu_i}(mk) - \mu_i - (\hat{\mu_1} - \mu_1) \geq \Delta_i] \end{aligned} \tag{2.7}$$

The next step would be to ensure that the terms, $\hat{\mu_i}(mk) - \mu_i - (\hat{\mu_1} - \mu_1)$ is *subgaussian*. In order to show that the term is a *subgaussian*, the following property[6, pp. 77–78] will be used.
Assume that $X_i - \mu$ are independent, $\sigma - subgaussian$ random variables. Then for any $\epsilon \geq 0$,

$$\mathbb{P}[\hat{\mu} \geq \mu + \epsilon] \leq exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) \qquad and \qquad \mathbb{P}[\hat{\mu} \leq \mu + \epsilon] \leq exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right) \tag{2.8}$$

Where $\hat{\mu} = \frac{1}{n}\sum_{t=1}^{n} X_t$, which is also known as the Empirical mean of X_t
As such, the following equation can be derived,

$$\mathbb{P}[\hat{\mu_i}(mk) - \mu_i - (\hat{\mu_1} - \mu_1) \leq exp\left(-\frac{m\Delta_i^2}{4}\right) \tag{2.9}$$

By subsituting Equation 2.9 and Equation 2.4 into Equation 1.3, Equation 2.2 can be obtained. From the bound illustrated in the regret analysis, it can be observed that if m is large, the bandit will tend to explore for too long and resulting in the first term being too large, on the contrary, if m is too small, the probability that the bandit exploits on the wrong arm will increase and resulting in the second term to be large. The dilemma of choosing an optimal value for m thus becomes crucial in order to ensure that the bandit has sufficient exploration and adequate exploitation. Assume a bandit with arm, k=2, and that the first arm is optimal resulting in $\mu^* = \mu_1 = \max_i \mu_i$. The following regret can thus be obtained,

$$\begin{aligned} R_t &\leq m\Delta + (t - 2m)\Delta \; exp\left(-\frac{m\Delta^2}{4}\right) \\ &\leq m\Delta + (t)\Delta \; exp\left(-\frac{m\Delta^2}{4}\right) \end{aligned} \tag{2.10}$$

Where $\Delta_1 = \mu^* - \mu_1 = 0$ and $\Delta_2 = \mu^* - \mu_2 = \mu_1 - \mu_2 = \Delta$. For large n the quantity on the right-hand side of Equation 2.10 is minimised up to a possible rounding error by,

$$m = max\left\{1, \quad \left\lceil \frac{4}{\Delta^2} \log\left(\frac{t\Delta^2}{4}\right) \right\rceil\right\} \tag{2.11}$$

2.1.2 Epsilon - Greedy Bandit

ϵ-greedy algorithm solves the performance issue faced in the *ETC* algorithm by spreading the exploration more uniformly over time. The term greedy comes from choosing only the best option in the given frame. In the ϵ-greedy algorithm, the exploration is also uniform over the number of arms. Still, since the exploration is now spread uniformly over time, it will be possible to obtain a much better regret even at a short time, t.
In a k-armed bandit problem, each individual action will generate a reward given the action that is selected. This is also called the value of the action. By denoting the action selected on each arm at a time, t as $Action_t$ and the respective reward as $Reward_t$. The value of the action of an unknown action, a denoted as $q(a)$.

$$q(a) = \mathbb{E}[Reward_t | Action_a = a] \tag{2.12}$$

As we do not know the exact action value, estimation could be performed on $q(a)$, which will be denoted as $q_{estimate}(a)$. This is estimated by averaging the reward received as $q(a)$ is the reward collected at action a.

$$q_{estimate}(a) = \frac{sums\ of\ reward\ that\ a\ taken\ prior\ to\ t}{number\ of\ times\ that\ a\ taken\ prior\ to\ t} \tag{2.13}$$

By selecting the greatest or the best optimal value based on the action would often be deemed as a greedy action which is the same as *exploiting* with the knowledge of knowing which result is the best. On the contrary, selecting any other random action would be known as *exploring*, which allows the algorithm to estimate the random action better than the optimal action.

$$Greedy\ (Exploitation): Action_t = \arg\max_t(q_{estimate}(a))$$
$$Non\text{-}Greedy\ (Exploration): Action_t = random(q_{estimate}) \ \in \ [0, 1]$$

It would be trivial to always opt for exploitation to maximize the reward. Still, in the long run, exploration may instead produce a much better reward as better action might be discovered during exploration, and the better reward could then be exploited to generate a better overall reward value. Equation 2.13 can be further simplified in terms of the total amount of incremental n counter actions step.

$$q_{estimate}(a) = Q_t = \frac{X_1 + X_2 + \cdots + X_{n+1}}{n - 1}$$

To implement a new estimation based on the previous estimation, it is essential to ensure that all the existing reward values are recorded so that the new $q_{estimate}$ can be estimated as Q_{t+1}, which can also be seen as the **Empirical Mean** of the sum of the rewards. It can be observed from Equation 2.14 that the update rule will always be biased by its initial estimate however, such an bias will be gone once every action has been selected.

$$
\begin{aligned}
Q_{t+1} &= \frac{1}{t}\sum_{i=1}^{T} X_i \\
&= \frac{1}{t}\left(X_t + \sum_{i=1}^{T-1} X_i\right) \\
&= \frac{1}{t}\left(X_t + (t-1)\frac{1}{(t-1)}\sum_{i=1}^{T-1} X_i\right) \\
&= \frac{1}{t}(X_t + (t-1) \times Q_t) \\
&= Q_t + \frac{1}{t}(X_t - Q_t)
\end{aligned}
\tag{2.14}
$$

By combining the equation mentioned before, the epsilon greedy bandit algorithm as be defined as algorithm 3.

Algorithm 3: Epsilon Greedy Bandit

Parameters: K arms, T rounds, ϵ value, a-random action

for *each round*, $t \in [T]$ **do**

$$A_t = \begin{cases} \max Q_t, & \text{with probability, } 1-\epsilon \\ \text{any action(a)}, & \text{with probability, } \epsilon \end{cases}$$

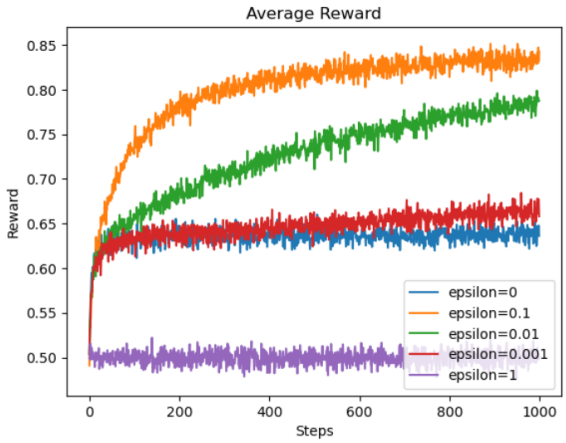
reward = $X_t(A_t)$

Update parameters:

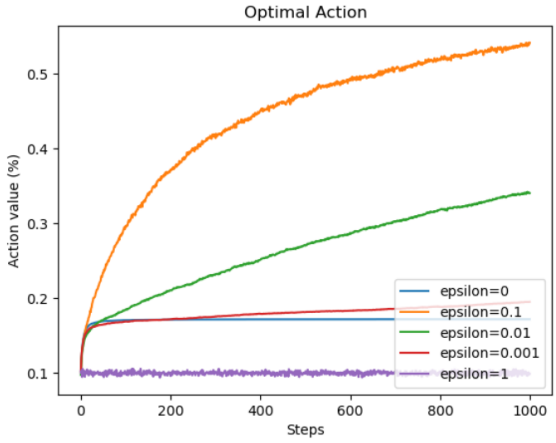
$Q_{t+1} = Q_t + \frac{1}{t} \times (reward - Q_t)$

end

An experiment was concluded with different values of ϵ in the set [0, 0.1, 0.001, 0.0001, 1] as shown in Figure 2.1. From the result, it can be seen that different value of ϵ gives a different variation of rewards. ϵ = 0 and ϵ = 1 produced distinct and expected results. As mentioned previously, due to it being a greedy algorithm, setting the ϵ to be "1" causes the algorithm to immediately exploit the reward, which causes the algorithm to have no possibility of any exploitation, resulting in a very bad reward generation as it is always trying to explore to a better reward. While having a ϵ = 0 causes the algorithm to be always "exploiting, " resulting in a rather consistent reward outcome. However, the shortcoming is obvious; as it is always exploiting, it will sometimes forgo the possibility of getting a high reward from exploration. This is apparent when comparing the reward between ϵ = 0 and ϵ = 0.1. This is the main challenge when using a greedy algorithm, as no one will know the perfect value for ϵ, and it will solely depend on the type of algorithm it is applied to.



(a) Average reward



(b) Optimal reward

Figure 2.1: Reward with varying epsilon values. Source: Github [7]

2.1.3 Optimistic Initial Bandit

Both subsection 2.1.1 and algorithm 3 are examples of a non-adaptive bandit. A non-adaptive bandit is a bandit that does not have any adaption in the exploration phase. The result of the observed rewards will not affect the exploration schedule. Both methods are dependent on the initial action estimate, $Q_t(a)$. In terms of statistical context, these methods are considered "biased" by the initial estimation. For the update rule shown in Equation 2.14, the bias disappears once all actions have been selected once. Still, for a method with constant step size, α the bias is permanent. An example of a new incremental update rule for the parameters can be shown below;

$$Q_{t+1} = Q_t + \alpha(R_t - Q_t) \tag{2.15}$$

Where the step size parameter, $\alpha \in (0, 1]$. This allows the update rule for Q_{t+1} to be a weighted average of the previous rewards and the initial estimate $Q_{t=1}$. As such, the new

increment update rules are as follows;

$$\begin{aligned} Q_{t+1} &= Q_t + \alpha(X_t - Q_t) \\ &= \alpha(X_t) + (1-\alpha)Q_t \\ &= \alpha(X_t) + (1-\alpha)[\alpha(X_{t-1}] + (1-\alpha)Q_{t-1}) \\ &= \alpha(X_t) + (1-\alpha)[\alpha(X_{t-1})] + (1-\alpha)^2 Q_{t-1} \\ &= \alpha(X_t) + (1-\alpha)[\alpha(X_{t-1})] + (1-\alpha)^2 Q_{t-2} + \\ &\qquad \cdots + (1-\alpha)^{t-1}[\alpha(X_1)] + (1-\alpha)^t Q_1 \\ &= (1-\alpha)^t Q_1 + \sum_{i=1}^{T} \alpha(1-\alpha)^{t-i} X_i \end{aligned} \tag{2.16}$$

As seen from Equation 2.16, it can be seen that with constant, α the initial action, $Q_{t=1}$ could be changed to encourage explorations instead of setting the initial value to zero. Depending on the initial value of $Q_{t=1}$, it could result in a better exploration consistency. With a larger $Q_{t=1}$, the reward will always seem lesser than the initial estimate. This allows the algorithm to always explore for better rewards even if a greedy result is always selected.
The following algorithm uses an epsilon-greedy bandit from algorithm 3 as the baseline while starting with a *initial estimate of* ***c*=5**.

Algorithm 4: Optimistic Initial Bandit

Parameters: K arms, T rounds, ϵ value, a-random action, $Q_{t=1} = \mathbf{c}$
for *each round*, $t \in [T]$ **do**

$$A_t = \begin{cases} \max Q_t, & \text{with probability, } 1-\epsilon \\ \text{any action(a)}, & \text{with probability, } \epsilon \end{cases}$$

reward = $X_t(A_t)$

Update parameters:
$Q_{t+1} = Q_t + \alpha(X_t - Q_t)$
end

Figure 2.2 shows the performances between an optimized initial bandit with an epsilon-greedy bandit. The experiment conducted previously(Figure 2.1) showed that an epsilon-greedy algorithm with $\epsilon = 0.1$ gives the best result. The comparison is only performed for $\epsilon = 0.1$ instead of other epsilon values. From Figure 2.2, it can be seen that the reward generated with a high initial value is much more optimal than the reward generated with a zero initial value. As the rewards are generated from a normal distribution with *mean* = 0 and *variance* = 1, an initial value of 5 is highly optimistic. Such a high initial value causes the algorithm to explore more. Initially, the optimistic method performed much worse as it explored more. Still, eventually, it outperformed the result generated by the bandit with zero initial value, as its exploration would decrease over time, as mentioned before. However, this algorithm is only effective for *stationary MAB* problems [5, pp. 34–35]. It will not be optimal for *non-stationary* problem as the window of exploration for the optimized initial bandit is only at the initial stage. If the task is changed, creating a new requirement for new

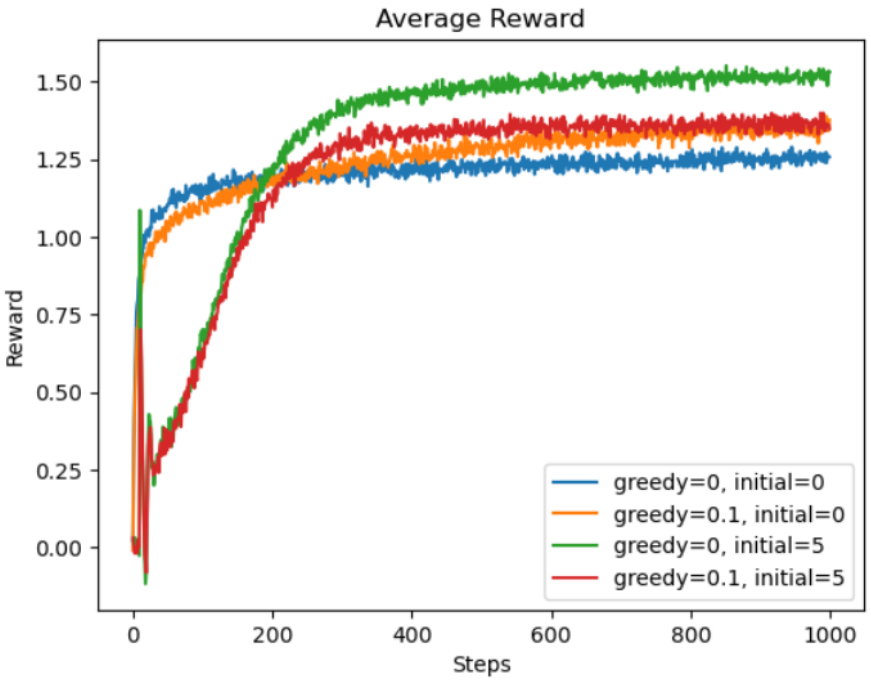


Figure 2.2: Result of Optimised Initial Bandit vs Epsilon-Greedy algorithm. Source: Github [7]

exploration will be impossible. Any method that focuses on the initial condition will be most unlikely to help with any general *non-stationary* problem as the beginning of time only occurs once.

2.1.4 Upper-Confidence Bound Bandit

Exploration is needed because there is always uncertainty about the accuracy of the action-value estimate. The greedy actions look best, but some other actions may be better. ϵ-greedy action selection forces the non-greedy action to be tried indiscriminately, with no preference for those nearly greedy or particularly uncertain. It would be better to select among the non-greedy action according to their potential for actually being optimal, considering both how close their estimate is to being maximal and the uncertainty in those estimates. One effective way of selecting such action is as follows;

$$A_t = \arg\max_a \left(Q_t(a) + c\sqrt{\frac{\ln t}{T_t(a)}} \right) \tag{2.17}$$

c denotes the confidence level, $N_t(a)$ denotes the number of times that action a has been selected prior to time t.
The idea of this UCB action selection is that the square-root term measures uncertainty or variance in the estimate of a action. The value will be maximized over and thus an upper

bound on the possible true value of action a with c being the confidence level. Each time an action other than a is selected, t increases but $N_t(a)$ does not. Using the natural log means the increment gets smaller over time but remains unbounded. Eventually, once every action has been selected, the action with a lower value estimate will be selected with decreasing frequency over time. However, the UCB algorithm would not be optimal for problems that are non-stationary [5, p. 36]. A non-stationary problem is a problem that has an optimal solution, but the optimal solution might change over time. It would require a much more complex algorithm to solve a non-stationary problem, such as the algorithm shown in **??**.

Algorithm 5: UCB

Parameters: K arms, T rounds, a-random action, α step size
for *each round*, $t \in T$ **do**
  for *each arm*, $a \in K$ **do**
    $A_a = \left(Q_t(a) + c\sqrt{\frac{\ln t}{T_t(a)}}\right)$
  end
  $A_t = \underset{a}{argmax}\,(A_a)$
  reward $= X_t(A_t)$

  Update parameters:
  $Q_{t+1} = Q_t + \alpha \times (reward - Q_t)$
end

The result with the comparison between UCB and Epsilon-Greedy algorithm are shown in Figure 2.3. The test was conducted on a 10-armed test bench. As seen in the figure, UCB results are generally better than Epsilon-Greedy, as shown in the figure. The UCB algorithm with $c = 0$ and $c = 2$ offers exciting results. With $c = 0$, it can be seen that no exploration has been done, and the bandit exploited the rewards, resulting in a linear and constant reward generated. While a UCB algorithm with $c = 2$ shows a significant fluctuation in the initial stages but as time passes, the algorithm would slowly converge,s and a much more stable result is obtained.

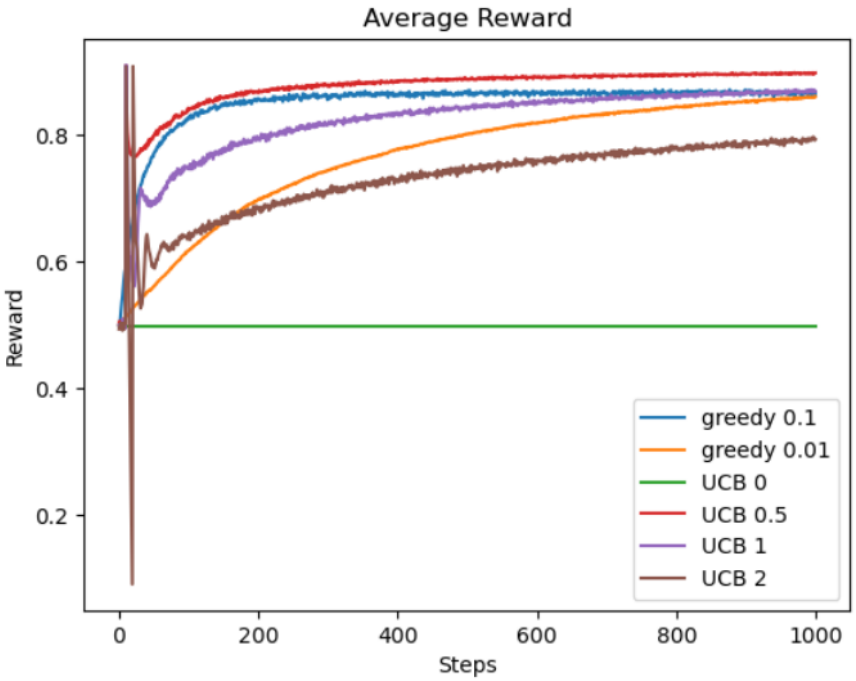


Figure 2.3: Result of UCB vs Epsilon-Greedy algorithm. Source: Github [7]

2.1.5 Kullback Leibler-UCB

A Kullback-Leibler-UCB (KL-UCB) algorithm utilizes the KL-Divergence to quantify how much one probability distribution differs from another. In a problem with a true distribution value and an approximation of that distribution value, it is helpful to use KL divergence to quantify the differences between the two distributions. Generally, this is also referred to as a problem of calculating the statistical distance between two probability distributions. As such, it is common instead to calculate the divergence between the two probability distributions. A divergence is like a measurement, but it is not symmetrical. This means that divergence is a measurement of how one distribution differs from another, where calculating the divergence for distributions P and Q would give a different score from Q and P.
The KL divergence between two distributions P and Q on the same sample space, χ, and the relative entropy from Q to P is shown as below;

$$\begin{aligned} KL(P||Q) &= -\sum_{x\in\chi} P(x) * \log\left(\frac{Q(x)}{P(x)}\right) \\ &= \sum_{x\in\chi} P(x) * \log\left(\frac{P(x)}{Q(x)}\right) \end{aligned} \tag{2.18}$$

Where $||$ denotes P divergence from Q. It is also important to remember that the divergence is not symmetrical.

$$KL(P||Q) \neq KL(Q||P)$$

In previous chapters, we assumed that the rewards was Gaussian for some known $\sigma > 0$. This has the advantage of simplicity and relative generality. However, stronger assumptions are sometimes justified and often lead to stronger results. For KL-UCB the rewards are assumed to be Bernoulli, which just means that $R_t \in [0,1]$. For such a Bernoulli bandit, in order to construct the confidence bounds for the unknown parameters, it is therefore crucial to analyse the concentration of the empirical mean for sums of Bernoulli random variables. As such, the relative entropy or KL-divergence for the Bernoulli distribution with parameters $p, q \in [0,1]$ are as follows;

$$d(p,q) = p\log(\frac{p}{q}) + (1-p)\log(\frac{1-p}{(1-q}) \tag{2.19}$$

Where the limits can be seen as;

$$d(p,q)|_{(p,q)\in[0,1]} = \begin{cases} \log(\frac{1}{1-q}), & \text{if } p = 0 \\ \log(\frac{1}{q}), & \text{if } q = 0 \\ 0, & \text{if } p = q = 0 \\ 1, & \text{if } p = q = 1 \\ \infty, & \text{otherwise} \end{cases}$$

The KL function, d(p,•) and d(•,q) can be seen as convex and therefore have unique minimised value at q and p receptively. This can be show by the prove as shown;

> Assume: (p,q) ∈ (0,1). d(•,q) is the sum of the negative binary entropy function denoted by h(p). $h(p) = p\log p + (1-p)\log(1-p)$. h(p) is also a linear function in which the second derivative of $h(p) = h^{''}(p) = \frac{1}{p} + \frac{1}{1-p}$ which is positive, hence h(p) is convex. For d(p,•), it is the sum of h(p) and the convex functions $p\log\frac{1}{q}$ and $(1-p)\log\frac{1}{1-q}$, hence d(p,•) is also convex.

For KL-UCB, the action value,A_t will be updated by obtaining the argmax of the maximised estimated value, q. In which the KL-divergence will be bounded by a function $f(t)$ which will be divided by the action count of each arms.

$$A_t = \underset{t}{argmax}[\underset{q}{max}\{q \in [0,1] : d(\hat{\mu}_i(t-1), q) \leq \frac{c\log f(t)}{T_i(t-1)}\}] \tag{2.20}$$

Where $f(t) = 1 + t\log^2(t)$. The empirical mean of the Bernoulli random variables can be calculated by;

$$\hat{\mu}_i = \frac{1}{n}\sum_{i=1}^{n} R_i \tag{2.21}$$

Algorithm 6: KL-UCB

for *each round*, $t \in T$ **do**
  for *each arm*, $i \in K$-*arms* **do**
    Calculate Empirical Mean
    $A_t = \underset{t}{argmax}[\underset{q}{max}\{q \in [0,1] : d(\hat{\mu}_i(t-1), q) \leq \frac{c \log f(t)}{T_i(t-1)}\}]$
  end
  reward = $X_t(A_t)$

  Update parameters:
  Action count, T_i of each arm is updated accordingly
end

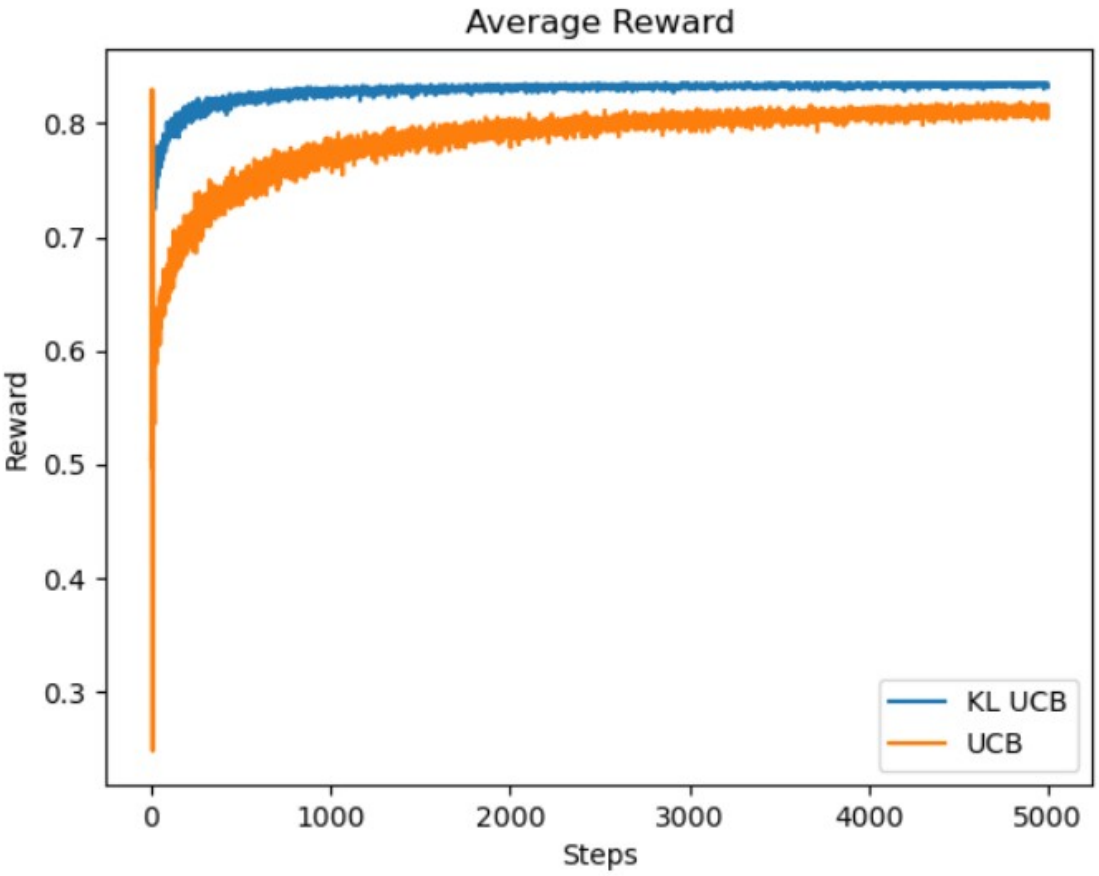


Figure 2.4: Result of KL UCB algorithm Source: Github [7]

2.1.6 Gradient Bandit

A Gradient bandit uses a gradient ascent algorithm to observe the respective reward. Gradient ascent works similarly to the famous and widely used gradient descent [8], with one difference. The task fulfills it not minimization but rather a maximization of some function. The reason for the difference is that, at times, we may want to reach the maximum, not the minimum, of some function. In this case, we would want to find the maximum reward of the algorithm instead of the lowest reward. Gradient ascent finds use in Logistic Regression Bandit. It is a supervised Machine Learning algorithm commonly used for prediction problems. It estimates the probability of an event occurring based on a given data set of independent variables by computing the log odds for the event as a linear combination of one or more independent variables. The main difference between gradient ascent and gradient descent is the goal of the optimization. In gradient ascent, the goal is to maximize the function, while in gradient descent, the goal is to minimize the function. As a result, while both rely on the gradient of the function to get the direction of the steepest slope, gradient ascent takes steps in the direction of the steepest slope, while gradient descent takes steps in the opposite direction. As a result, the iterative update rule for each step of the algorithm thus becomes;

$$Q_{t+1}(A_t) = Q_t(A_t) + \alpha(X_t - \overline{X}_t)(1 - \pi_t(A_T)) \tag{2.22}$$

α denotes the step-size parameter and is always larger than "0". $\overline{R}_t$ is the average of all the rewards up to the current time, t. Q_t is initialized such that all actions have an equal probability of being selected such that;

$$Q_{t=1}(a) = 0 \quad \forall \quad a$$

$\pi_t(A_t)$ denotes the probability of taking action a at a given time t. It is usually determined by a soft-max distribution such as;

$$\pi_t(A_t) = \frac{e^{Q_t(a)}}{\sum_{b=1}^{n} e^{Q_t(b)}}$$

$\overline{X}_t$ also serves as the baseline in which the rewards are compared. If the reward is higher than the baseline, the probability of taking A_t in the subsequent runs will be increased. And if the reward is lower than the baseline, the probability will be decreased. Gradient bandit can also be understood as a stochastic approximation to gradient ascent [5, pp. 38–40].

Algorithm 7: Gradient Ascent

Parameters: K arms, T rounds, N steps, a-random action, α step size

for *each round*, $t \in T$ **do**

$A_t = \pi_t(A_t) = \frac{e^{Q_t(a)}}{\sum_{b=1}^{n} e^{Q_t(b)}}$

reward = $X_t(A_t)$

Update parameters:

$Q_{t+1} = Q_t + \alpha \times (reward - Q_t)$

end

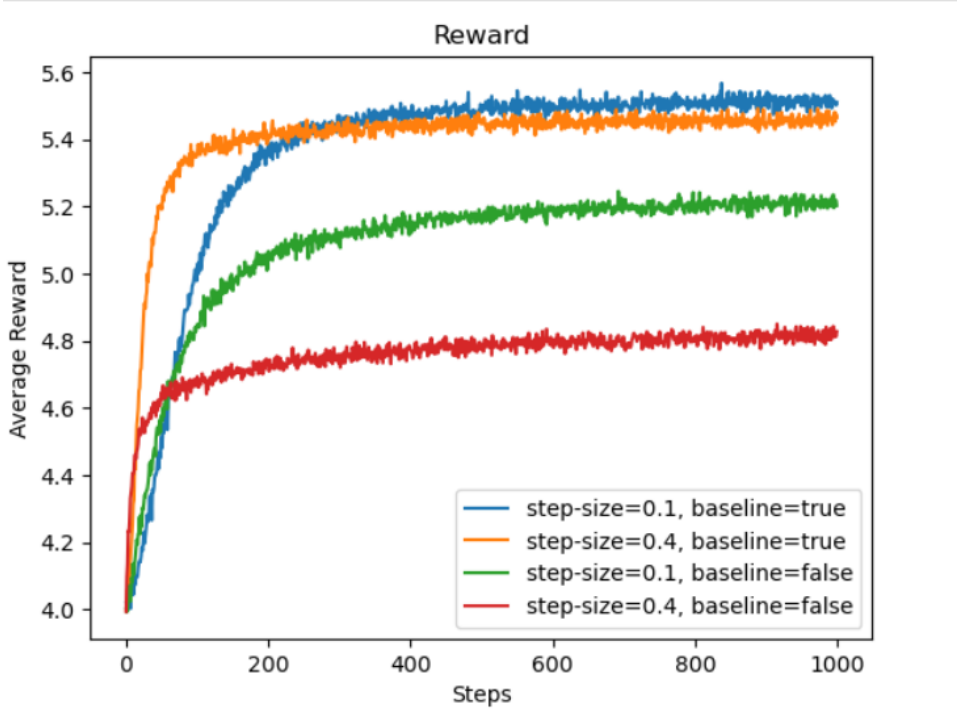


Figure 2.5: Result of Gradient Ascent algorithm. Source: Github [7]

Figure 2.5 shows the result of the experiment between different gradient ascent algorithms with varying settings. It can be observed that the rewards with a baseline generated much better than the algorithm without a baseline. This shows that if the baseline was removed($\overline{R}_t$ was treated as zero), resulting in Equation 2.22 to be simplify to the following;

$$Q_{t+1}(A_t) = Q_t(A_t) + \alpha(X_t)(1 - \pi_t(A_T)) \tag{2.23}$$

It can also be observed that with a different value for the step-size parameter, α, a different result was obtained. But it was obvious that the step size of 0.1 performed better than with step size of 0.4. This became even more significant when the baseline, $\overline{X}_t$ was removed.

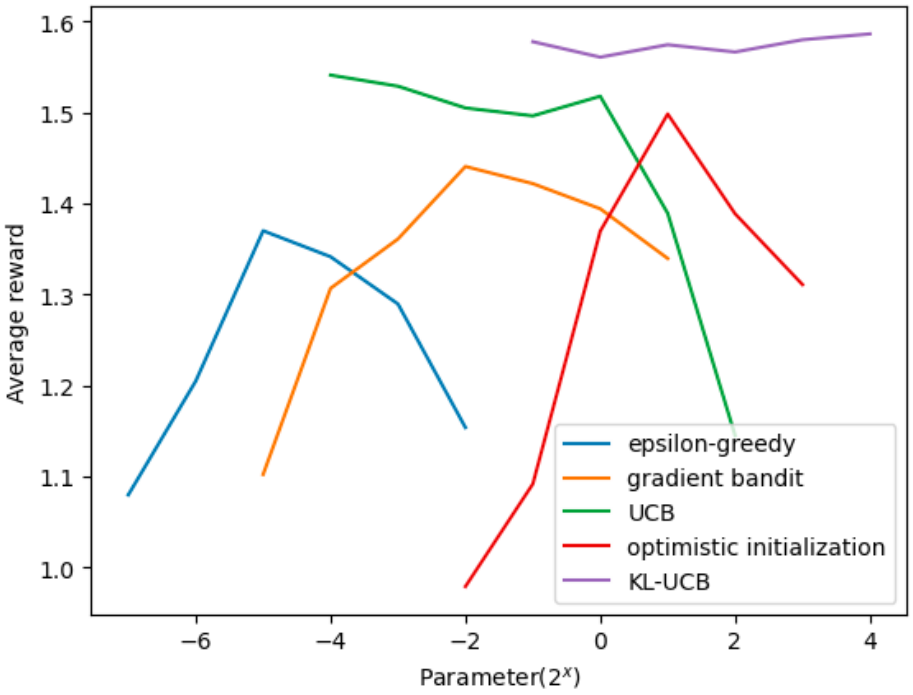


Figure 2.6: Result of comparing all the Stochastic bandit with varying value based on the individual bandit.

An experiment was conducted with all the stochastic bandit that was discussed. Figure 2.6 shows the reward generated by each stochastic bandit with varying parameter that is unique to each bandit. Such as, $\epsilon, \alpha, c, initial\_value, c$ for epsilon greedy, gradient, UCB, optimistic initial and KL-UCB respectively. From the plot, it can be easily noticed that all the algorithm performs best at an intermediate value of their parameters. And KL-UCB is able to achieve the best and is least sensitive to the parameter as well. Although different value for the parameter was shown, it is still difficult to accurately determine what is the perfect or preferred value for the parameter as the value will have to differ accordingly to the complexity of the bandit involved.

2.2 Bayesian Bandit

The Bayesian bandit problem adds the Bayesian assumption[9, p. 16] to stochastic bandits: the problem instance I is drawn initially from some known distribution P. The time horizon T and the number of arms K are fixed. Then an instance of stochastic bandits is specified by the mean reward vector and the reward distributions. The distribution P is called the prior distribution or the Bayesian prior. The goal is to optimize Bayesian regret.
Using the classic slot machine example, with a pull of the slot machine, the result obtained will be denoted as x, which is a Bernoulli random variable. The distribution can be described as follows;

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{2.24}$$

Where $P(\theta|x)$, $P(x|\theta)$, $P(\theta)$, $P(x)$ denotes the Posterior, Likelihood, Prior and Evidence respectively. Using the Bayesian analysis method, with more additional data gathered during exploration during the experiments, will result in the posterior distribution getting closer to the ideal targeted distribution. Through this process, it will be able to estimate better the parameter, θ, which is also known as the action's success probability or mean reward. As the variable x is a Bernoulli random variable, the natural distribution choice would be a binomial distribution. As more pull of level of the slot machine is executed, the number of x will start to accumulate. And this is essentially the same as sampling from a binomial distribution with s success and f failure out of a total of $(s + f)$ trials. Such a binomial distribution can be shown as follows;

$$P(s, f|\theta) = \binom{s+f}{s}\theta^s(1-\theta)^f \tag{2.25}$$

Initially, it might be beneficial to use a uniform prior distribution, but having a uniform prior would only make sense on the very first try; there will not be any historical data to estimate the parameter from. It is also worth noticing that a prior is a **conjugate** for the likelihood function if and only if the posterior is of the same type as the prior[10]. The relationship can be summarized and shown in Table 2.1.

Likelihood	Conjugate Prior	Posterior
Bernoulli	Beta	Beta
Binomial	Beta	Beta

Table 2.1: Choice of conjugate prior and posterior for respective likelihood

As a result, Beta distribution is a very convenient choice of priors for Bernoulli rewards.

The PDF and mean of beta distribution, $Beta(\alpha, \beta)$ with $\alpha > 0$ and $\beta > 0$ is shown as below;

$$f(\theta; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

$$\mathbb{E}[\theta] : \frac{\alpha}{\alpha + \beta} \qquad \mathbb{V}[\theta] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

Where Γ denotes the Gamma function.
Let $x \in 0, 1$ be distributed according to Bernoulli distribution with parameter, $\theta \in [0, 1]$ such that $p(x|\theta) = \theta^x(1-\theta)^{1-x}$. And θ be distributed according to Beta distribution with parameter, (α, β) such that $p(\theta|\alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$. By multiplying both Bernoulli and beta distribution, we can obtain the following equation,

$$\begin{aligned} p(\theta|x, \alpha, \beta) &\propto p(x|\theta)p(\theta|\alpha, \beta) \\ &\propto \theta^x(1-\theta)^{1-x}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \theta^{\alpha+x-1}(1-\theta)^{\beta+(1-x)-1} \\ &\propto p(\theta|\alpha + x,\ \beta + (1-x)) \end{aligned}$$

It is worth noticing that the last line of the equation is actually the beta distribution with parameter $(\alpha + x, \beta + (1-x)$. And this will be an essential equation which will form the new update rule for the Bayesian bandits. As more and more observations are obtained, the parameter will be updated as follows;

$$(\alpha_{t+1},\ \beta_{t+1}) = (\alpha_t + reward,\ \beta_t + (1 - reward))$$

The beta distribution is helpful for Bernoulli rewards because if the prior is a Beta(α, β) distribution, then after observing a Bernoulli trial, the posterior distribution is simply Beta$(\alpha + 1, \beta)$ or Beta$(\alpha, \beta + 1)$, depending on whether the trial resulted in success or failure, respectively. Therefore, α is also known as the success prior, while β is known as the failure prior.

2.2.1 Thompson Sampling

Thompson sampling is also known as posterior sampling and probability matching, where actions are taken sequentially to balance exploiting what is known to maximize immediate performance and investing to accumulate new information that may improve future performance[11]. The algorithm efficiently addresses various problems computationally and enjoys wide use. A generalization of Thompson sampling to arbitrary dynamical environments and causal structures has been shown to be the optimal solution to the adaptive coding problem with actions and observations. In Thompson sampling, an action value is conceptualized as a sum over a set of action values. As the algorithm proceeds, it learns the abstract properties. It adopts the action value that minimizes the relative entropy to the action value with the best prediction of the bandit's action value. A simple intuition of how Thompson sampling is

explained as follows;

1) Initially, all machines are assumed to have a uniform distribution of the probability of success, which, in this scenario, is getting a reward.

2) For each observation obtained from the slot machine, a new distribution is generated with probabilities of success for each slot machine based on the reward.

3) Additional observations are established on these prior probabilities obtained on individual round or observation, which will then updates the success distributions.

4) After sufficient observations, each slot machine will have a successful distribution associated with it which can help the player choose the machines wisely to get the maximum rewards

2.2.2 General-Thompson Sampling

For simplicity of discussion, the following Thompson Sampling algorithm will be utilizing the Bernoulli bandit problem[11, p 20], i.e., when the rewards are either 0 or 1 and for arm i the probability of success (reward =1) is μ_i. With a reward distribution in the range of $[0, 1]$. The algorithm for Bernoulli bandits maintains Bayesian priors on the Bernoulli means μ_i. The Thompson sampling initially assumed that arm, i has prior Beta(1, 1) on μ_i, which is essential and natural as Beta(1, 1) forms the uniform distribution on (0,1). The parameters (α_k, β_k) are sometimes called pseudo-counts since α_k or β_k increases by one with each observed success or failure, respectively. A beta distribution with parameters Beta(α_k,β_k) has mean $\frac{\alpha_k}{(\alpha_k+\beta_k)}$, and the distribution becomes more concentrated as $\alpha_k + \beta_k$ grows.

Algorithm 8: General Thompson sampling algorithm

for *each round, t = 1, 2, ..., T* **do**
  for *k = 1, ..., K-arms* **do**
    $\hat{\theta_k} = sample\ beta(\alpha_t, \beta_t)$
  end
  $A_t = \arg\max_k \left(\hat{\theta_k}\right)$
  reward = $X_t(A_t)$

  Update parameters:
  $(\alpha_{t+1}, \beta_{t+1}) = (\alpha_{action_t} + reward,\ \beta_{action_t} + (1 - reward))$
end

2.2.3 Greedy-Thompson Sampling

The General-Thompson sampling algorithm could also be combined with a greedy approach. Greedy algorithms may be the simplest and most common approach to online decision problems. Such an algorithm is greedy because an action is chosen solely to maximize immediate reward. As previously mentioned at subsection 2.1.2, this allows the Thompson sampling to have a quick and simple approach to a result. A Greedy-Thompson sampling could be summarized in the following two steps;

1. *Estimate a model from historical data.*
2. *Select the action that is optimal for the estimated model.*

Similarly to the greedy bandit, the shortcoming of Greedy TS is identical. The algorithm might not actively explore or exploit due to the selection of the ϵ value, and this might cause the performance to be slightly affected. For a greedy Thompson sampling given each time period, t the algorithm generates an estimate, $\hat{\theta}_k = \frac{\alpha_k}{\alpha_k+\beta_k}$ which can be observed to be the same as the mean of the Beta distribution, Beta(α, β) mentioned in subsection 2.2.2. The estimate will equal its current expectation of the success probability, θ_k. The action with the largest estimate, $\hat{\theta}_k$ is then applied, and a reward will be observed based on this action. After which, the parameters of both α and β will be updated. Both algorithms are highly similar, as seen from both algorithm 8 and algorithm 9. The only difference is that the success probability estimate $\hat{\theta}_k$ is calculated by taking the mean of the distribution parameter (α, β).

Algorithm 9: Greedy Thompson sampling algorithm

for *each round, t = 1, 2, ..., T* **do**
  for *k = 1, ..., K-arms* **do**
    $\hat{\theta}_k = \frac{\alpha}{\alpha+\beta}$
  end
  $A_t = \arg\max_k \left(\hat{\theta}_k\right)$
  reward = $X_t(A_t)$

  Update parameters:
  $(\alpha_{t+1}, \beta_{t+1}) = (\alpha_{action_t} + reward,\ \beta_{action_t} + (1 - reward))$
end

Figure 2.7 shows the experiment result between the General Thompson Sampling and Epsilon-Greedy Thompson Sampling. From the result it was very obvious that for with an epsilon value of 1, the result will not useful as the reward will always be exploited due to the high value of epsilon which is result obtained previously mentioned on the epsilon greedy bandit, as shown in 2.1a. However, an interesting result have been observed. Previously, in 2.1a it was shown that an epsilon value of 0.1 prove to be the most efficient choice for epsilon, but for the new greedy Thompson Sampling algorithm, it was shown that the epsilon

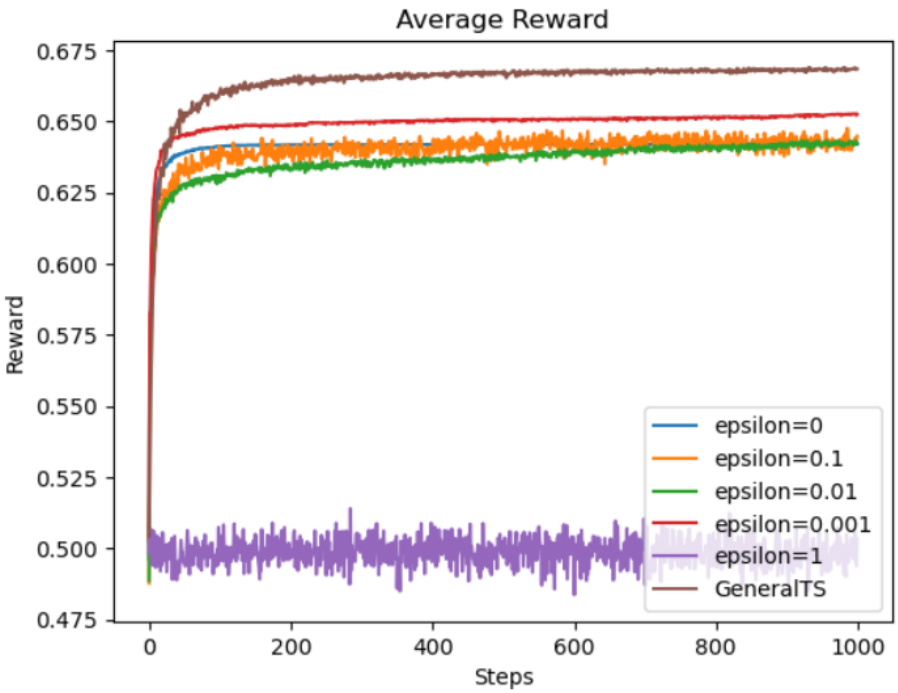


Figure 2.7: Result of GeneralTS vs Epsilon-Greedy TS algorithm. Source: Github [7]

value of 0.001 was able to deliver a much better result. It was also worth noticing that the General Thompson Sampling algorithm is able to provide a much better result compared to the different varying values of epsilons.

2.2.4 Laplace-Thompson Sampling

Laplace-Thompson sampling is a form of Thompson sampling that utilizes Laplace approximation for approximating a posterior distribution with a Gaussian centered at the maximum posterior estimate[12, p 303-312]. This method approximates the integral of a function $f(\theta)d\theta$ by fitting a Gaussian at the maximum $\hat{\theta}$ of $f(\theta)$ and computing the volume of the Gaussian [9, p 287]. The Hessian Matrix determines the covariance of the Gaussian, C of $log(f(\theta))$ at the maximum point $\hat{\theta}$. To find a Laplace approximation of $f(\theta)$ with a given second-order Taylor approximation of the log-density of function, $f(\theta)$

$$\ln(f(\theta)) \approx \ln(f(\overline{\theta})) - \frac{1}{2}(\theta - \overline{\theta})^T C(\theta - \overline{\theta})$$

Where

$$C = -\nabla^2 \ln(f(\overline{\theta}))$$

As an approximation to the density f, it was observed that

$$\tilde{f}(\theta) \propto \exp(-\frac{1}{2}(\theta - \overline{\theta})^T C(\theta - \overline{\theta}))$$

This is proportional to the density of a Gaussian distribution with mean $\overline{\theta}$ and covariance C^{-1}, and hence

$$\tilde{f}(\theta) = \sqrt{|\frac{C}{2\pi}|} exp(-\frac{1}{2}(\theta - \overline{\theta})^T C(\theta - \overline{\theta}))$$

With the approximation result, the algorithm for Laplace-Thompson sampling is shown below;

Algorithm 10: Laplace Thompson sampling algorithm

for *each round, t = 1, 2, ..., T* **do**
  for *k = 1, ..., K-arms* **do**
    Calculate $\tilde{f}(\theta)_k$
  end
  $action_t = \arg\max_k (\tilde{f}(\theta)_k)$
  reward = Bernoulli reward($action_t$)

  Update parameters:
  $(\alpha_{t+1}, \beta_{t+1}) = (\alpha_{action_t} + reward,\ \beta_{action_t} + (1 - reward))$
end

Figure 2.8 shows the result of the Laplace approximated Thompson Sampling and the epsilon greedy Thompson Sampling. It was observed that the Laplace algorithm was not able

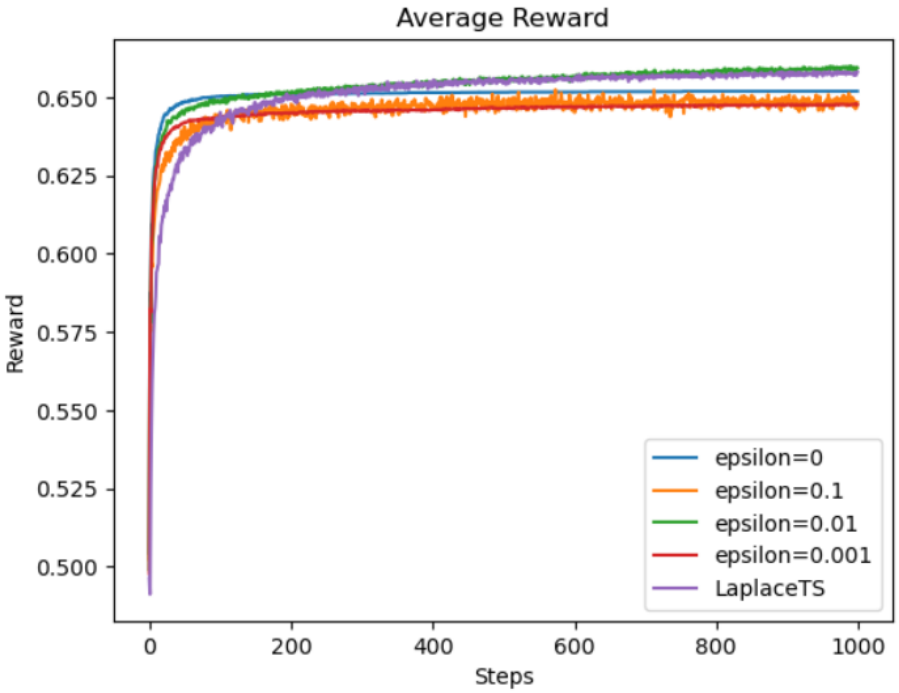


Figure 2.8: Result of LaplaceTS vs Epsilon-Greedy TS algorithm. Source: Github [7]

to achieve a better average reward compared to the typical greedy algorithm. One hypothesis to deduct what actually causes this might be the amount of iteration that was conducted for training of the experiment. As a greedy algorithm was discussed and proved that it was able to provide a better result in a shorter time due to it being able to exploit the reward at an early stage of the experiment. Therefore, with a much more complex approximation like the Laplace approximation, in the short term, the performance will not be as optimal as the greedy algorithm. As shown in Figure 2.9, it can be observed that with a much longer steps iteration used for the training in the experiment, the Laplace approximation was able to achieve a better average reward as compared to the greedy algorithm. In which it corresponds to the hypothesis made and shows that a Laplace approximation will in fact outperform a greedy algorithm given a longer step iterations.

2.2.5 Bootstrap-Thompson Sampling

The bootstrap method is a statistical technique for estimating quantities about a population by averaging estimates from multiple small data samples. Importantly, samples are constructed by drawing observations from a large data sample one at a time and returning them to the data sample after they have been chosen. This allows a given observation to be included in a small sample more than once. This sampling approach is also called sampling with replacement.
The basic idea of bootstrap is to make inferences about an estimate(such as sample mean)

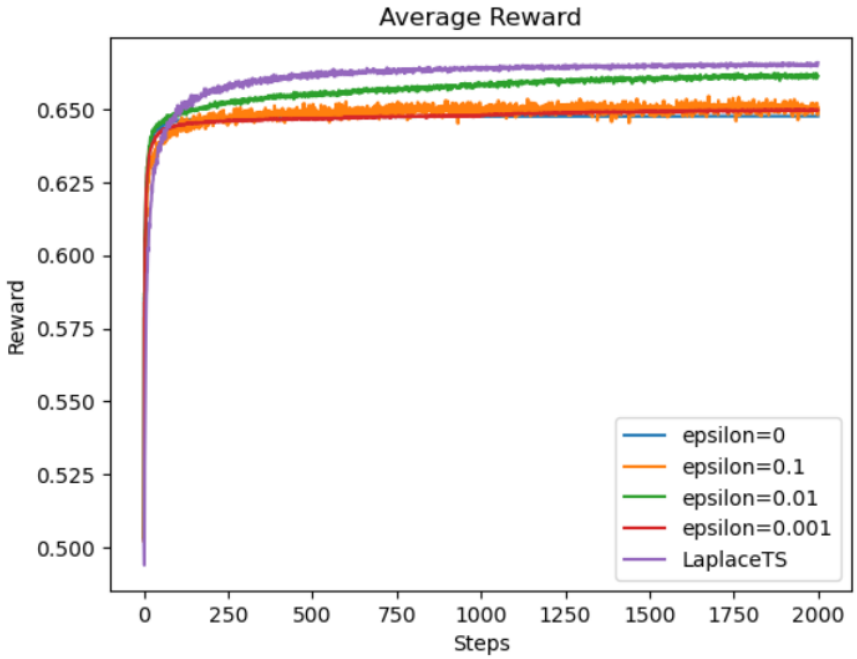


Figure 2.9: Result of LaplaceTS vs Epsilon-Greedy TS algorithm with longer step count. Source: Github [7]

for a population parameter, θ (such as the population means, $\tilde{\theta}$) on sample data. It is a re-sampling method by independent sampling with replacement from an existing sample data with the same sample size n and performing inference among these re-sampled data. Similarly to Laplace-Thompson sampling, the bootstrap method assumes that θ is drawn from a Euclidean space $\mathbb{R}^K$ The general concept of how the bootstrap method is shown below;

1. Draw a sample from the original sample data with replacement size n and replicate K times; each re-sampled sample is called a Bootstrap sample, and there is a total of K Bootstrap samples.

2. Evaluating the statistic of θ for each sample.

3. A sampling distribution with K Bootstraps statistic will be observed.

As shown above, Bootstrap provides an easy and simple way of getting a sampling distribution from a small data samples. It is a straightforward method to estimate the confidence interval for a complex distribution estimator like a MAB problem. Bootstrap can be seen as an effective way to control the stability of the results ad for most problems, it would be impossible to know the true confidence interval and is also a convenient method that avoids repeating the experiment to generate other sample data. This supports that Bootstrap sampling is a powerful and useful technique that allows one to approximate a probability

distribution from an unknown distribution.

Algorithm 11: Bootstrap Thompson sampling algorithm

for *each round*, $t = 1, 2, ..., T$ **do**
  for $k = 1, ..., K$-*arms* **do**
    Calculate Bootstrap distribution samples, θ_k
  end
  $action_t = \arg\max_k (\theta_k)$
  reward = Bernoulli reward($action_t$)

  Update parameters:
  $(\alpha_{t+1}, \beta_{t+1}) = (\alpha_{action_t} + reward,\ \beta_{action_t} + (1 - reward))$
end

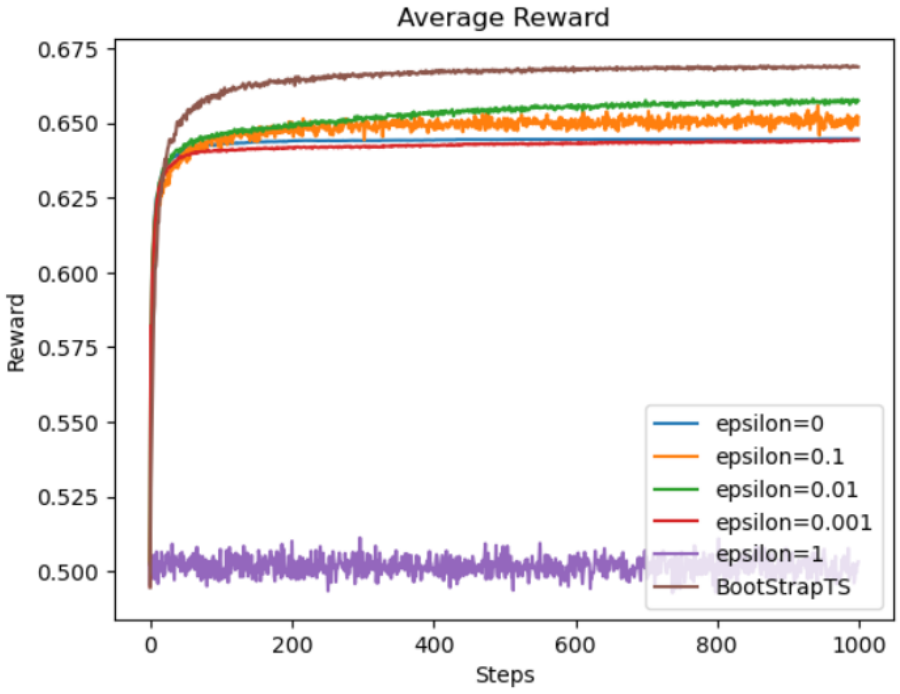


Figure 2.10: Result of BootstrapTS vs Epsilon-Greedy TS algorithm. Source: Github [7]

Figure 2.10 shows the result of the Bootstrap algorithm and the greedy algorithm. Similarly to the result of General Thompson Sampling, the result of the bootstrap approximation was able to produce a much better result as compared to the different varying greedy algorithm. However, it was worth noting that the computation time for the bootstrap approximation was much longer as compared to the greedy algorithm(About 1.6 times longer).

2.2.6 Langevin-Thompson Sampling

Langevin Monte Carlo Thompson Sampling[13] uses Markov Chain Monte Carlo methods to sample from the posterior distribution in contextual bandits directly. It is a much more computationally efficient method than Laplace-approximation since it only needs to perform noisy gradient descent updates without constructing the Laplace approximation of the posterior distribution. It can also learn the exact posterior distribution of parameter θ up to high precision. As stated in the paper in *Langevin Monte Carlo for Contextual Bandits*[14], two standard modifications to the typical Markov chain method to improve the computational efficiency. The first modification is to implement a stochastic gradient Langevin Monte Carlo, which uses sampled mini-batches of data to compute approximate rather than exact gradients. When fewer than 100 observations are available, the algorithm will follow the Markov chain with exact gradient computation. However, with more than 100 observations, it still follows the same previous procedure, but it uses an estimated gradient at each step based on a random sub-sample of the 100 data points. The second modification would involve the usage of a preconditioning matrix to improve the mixing rate of the Markov chain. To obtain good conditioning, an extremely small step size is required.
A Markov chain is shown below before the two modifications;

$$\theta_{t+1} = \theta_t + \epsilon \nabla \ln\left(f(\theta_t)\right) + \sqrt{2\epsilon} W_t \tag{2.26}$$

After the two mentioned modifications are applied, Equation 2.26 becomes;

$$\theta_{t+1} = \theta_t + \epsilon A \nabla \ln\left(f(\theta_t)\right) + \sqrt{2\epsilon} A^{\frac{1}{2}} W_t \tag{2.27}$$

The conditioning matrix A is defined by;

$$A = -(\nabla^2 \ln\left(f(\theta)\right))^{-1}$$

Algorithm 12: Langevin Thompson sampling algorithm

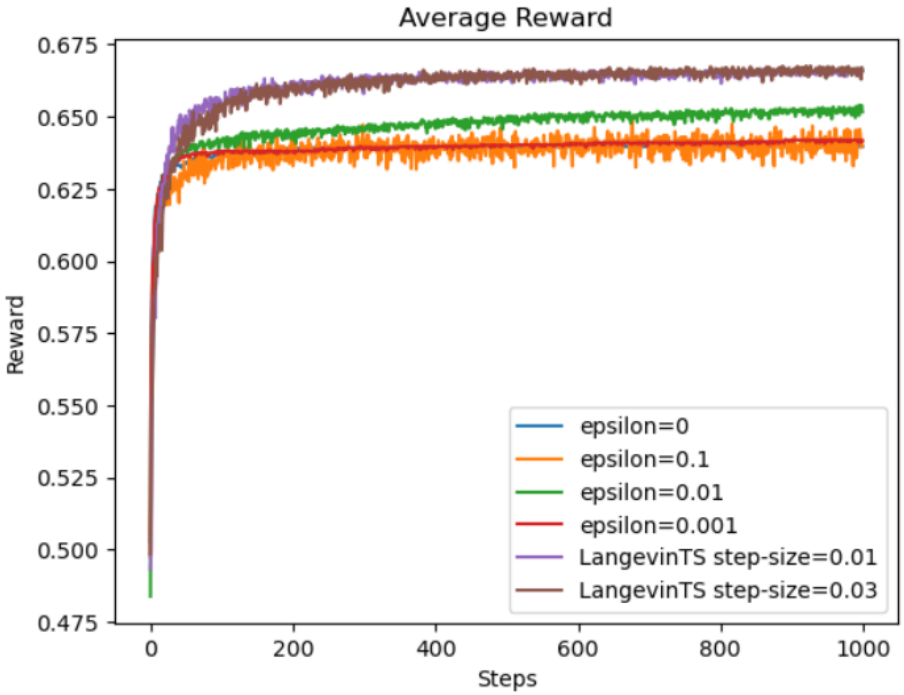
for *each round, t = 1, 2, ..., T* **do**
  for *k = 1, ..., K-arms* **do**
    Calculate conditioning matrix, A
    Calculate hessian of $\ln(f(\theta_t))$
    for *n = 1, ..., step-size* **do**
      Applying the conditioning matrix to the hessian and random Gaussian noise, W_n
      Calculate and observe θ_k
    end
  end
  $action_t = \arg\max_k(\theta_k)$
  reward = Bernoulli reward($action_t$)

  Update parameters:
  $(\alpha_{t+1}, \beta_{t+1}) = (\alpha_{action_t} + reward,\ \beta_{action_t} + (1 - reward))$
end

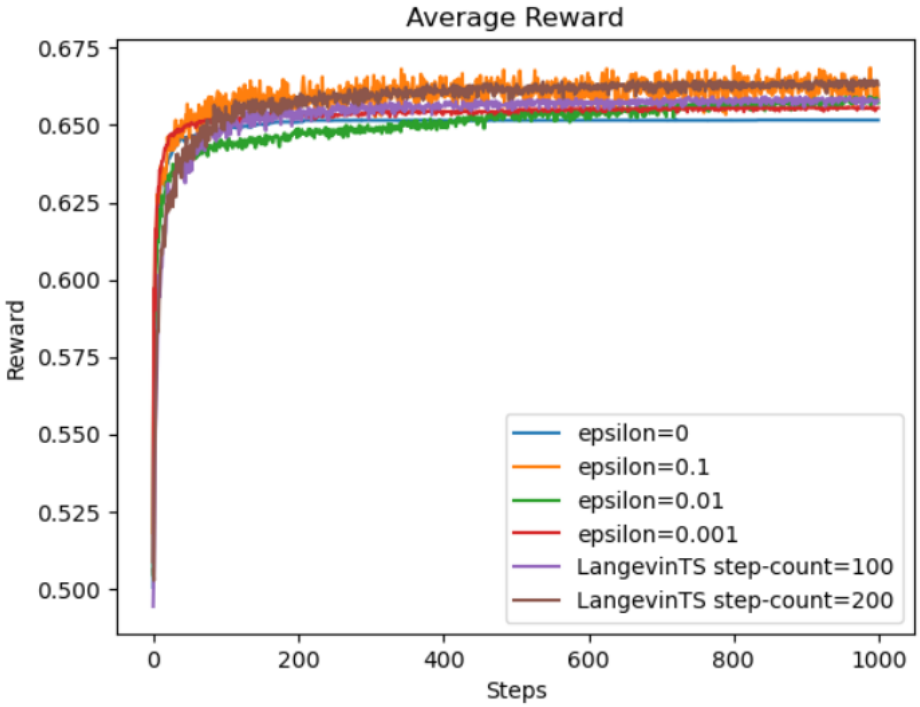
Figure 2.11 shows the result conducted for the experiment for Langevin algorithm. Firstly, an experiment was conducted to compare the effect of the step size parameter, ϵ. From 2.11a, it can be observed that the value of step-size does not affect much of the result. Both algorithm with different step size would still be able to generate similar reward and the trend of both algorithm is highly identical as well. Therefore, an additional experiment was conducted to observe the effect of the step count parameter instead. As shown in 2.11b, the results are much more apparent. Both step-size was kept the same so as to maintain fairness in the experiment and only the step count was adjusted. The reward generated by the algorithm with 200 step-count was observed to be much better then the algorithm with just 100 step-count. This implies that with a larger step count, the random sub-sample will be larger, and this will allow the algorithm to have a higher chance of exploiting better rewards. But due to the additional step count, it was also observed that the training time was increased significantly.

In conclusion, a experiment was then conducted to compare all the different type of Thompson sampling algorithm as shown in Figure 2.12. It can be seen that both the General Thompson sampling and Langevin algorithm was able to produce highly similar results. With Laplace approximation algorithm just barely behind the two mentioned algorithm. But it can be noticed that the Laplace method average reward still have the trend of further improving the average rewards, and as showed earlier at Figure 2.8, it can be can seen that with a much longer step iterations, Laplace approximation can eventually perform better. However, each individual algorithm have their own pros and cons. A balance will be required better the training time and also complexity of the problem faced. Although a greedy algorithm was often able to provide a decent optimal result in a short time, but for a much more complex problem or a problem that requires a much longer iteration time, a greedy algorithm might not be a wise choice. And vice versa for Laplace approximation, which will be redundant if

the MAB problem as a simple and straightforward problem.
For each of the methods that have been discussed, the computation time required per time period grows as time progresses. This is because each past observation must be accessed to generate the next action. This differs from default TS algorithms that was discussed earlier, which maintain parameters that encode a posterior distribution, and update these parameters over each time period based only on the most recent observation. In order to keep the computational burden manageable, it can be important to consider incremental variants of our approximation methods.



(a) Average reward with different step size. Source: Github [7]



(b) Average reward with different step count. Source: Github [7]

Figure 2.11: Average rewards for Langevin TS.

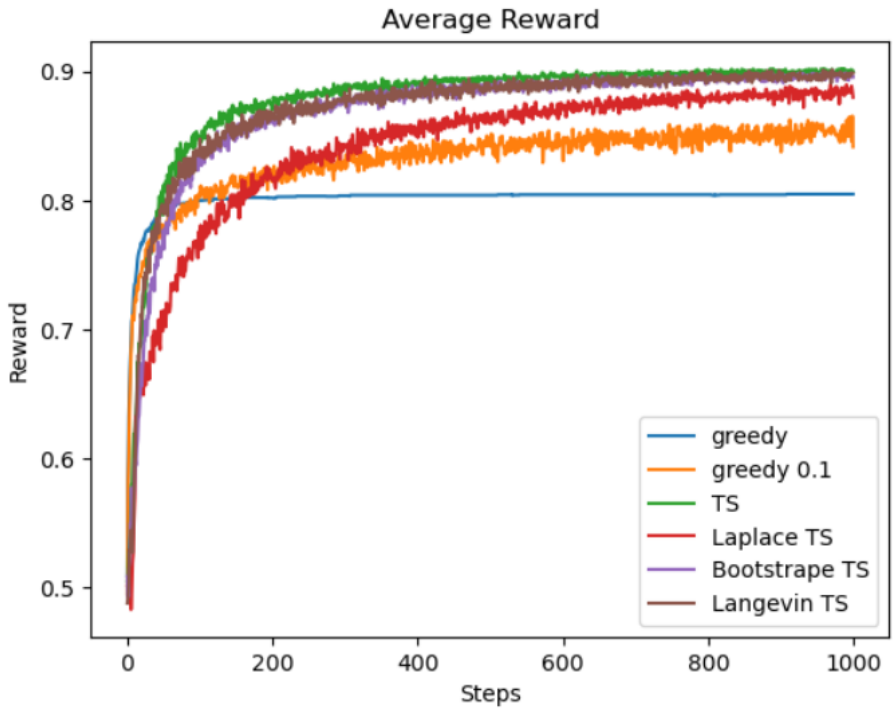


Figure 2.12: Result all the discussed TS algorithm. Source: Github [7]

3 Conclusion

Finally, with all the different variation of stochastic bandit and bayesian bandit that was discussed and implemented, interesting result can be obtained and observed. Although it seems that greedy options appear to be the worst algorithm but such an easy and straight forward algorithm is still widely used even till now. And it is essential to first study purpose of what the bandit will be used before selecting the most optimal bandit for the problem. If the purpose of the bandit is simply to determine a dynamic pricing system, a greedy approach will be the fastest and easily to implement and an optimistic result can be obtained. However, if the problem would to be to calculate the probability of a dynamic price being the optimal price, a slightly complicated bayesian bandit might need required as this would have to take into consideration the history information so as to predict a much more accurate result. Most of the figures that was shown in this report was mainly a comparison of just the reward of each algorithm. In the Github [7], a comparison between regret and the optimal reward can be found as well along with all the codes that was used to produce each individual figures. The next step forward would be to identify the possiblity of extending the research conducted thus far into an actual master thesis topic.

List of Figures

Glossary

mean refers to the average of a set of values. 12

non-stationary problems are those whose reward value is non-static and changes over time. 12–14

stationary problems are those whose reward value is static and does not change as time progresses. 12

variance refers to a statistical measurement of the spread between numbers in a data set. 12

Acronyms

MAB Multi-Armed Bandit. 1, 2, 6, 7, 12, 28, 32

UCB Upper Confidence Bound. 6

Bibliography

[1] A. Slivkins et al. "Introduction to multi-armed bandits". In: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), pp. 1–286.

[2] D. Bouneffouf, I. Rish, and C. Aggarwal. "Survey on Applications of Multi-Armed and Contextual Bandits". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185782.

[3] E. C. Jaya Kawale. "A Multi-Armed Bandit Framework for Recommendations at Netflix". In: 2018.

[4] V. Kuleshov and D. Precup. "Algorithms for multi-armed bandit problems". In: *arXiv preprint arXiv:1402.6028* (2014).

[5] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[7] L. C. K. Leong. *Research Internship on Multi-Armed Bandits*. https://github.com/LewisChua/ResearchInternship. 2023.

[8] P. Netrapalli. "Stochastic gradient descent and its variants in machine learning". In: *Journal of the Indian Institute of Science* 99.2 (2019), pp. 201–213.

[9] D. Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[10] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[11] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen, et al. "A tutorial on thompson sampling". In: *Foundations and Trends® in Machine Learning* 11.1 (2018), pp. 1–96.

[12] W. Penny, S. Kiebel, and K. Friston. *Variational bayes*. Elsevier, London, 2006.

[13] E. Mazumdar, A. Pacchiano, Y. Ma, M. Jordan, and P. Bartlett. "On approximate Thompson sampling with Langevin algorithms". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6797–6807.

[14] P. Xu, H. Zheng, E. V. Mazumdar, K. Azizzadenesheli, and A. Anandkumar. "Langevin monte carlo for contextual bandits". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 24830–24850.