Exploration vs Exploitation

Kushal Kharel

2023-10-31

The exploration-exploitation tradeoff encapsulates the delicate balance between venturing into the unknown and exploiting known resources or strategies. In numerous fields, from artificial intelligence to business strategy, individuals and systems confront the challenge of optimizing between the risks and rewards of exploration and exploitation. Exploration involves seeking new opportunities, fostering innovation, and embracing uncertainty, while exploitation centers on maximizing gains through proven methods and established knowledge. The key lies in finding the optimal equilibrium; too much exploration risks inefficiency and failure, while excessive exploitation may lead to missed opportunities and stagnation. Successful navigation of this tradeoff is essential for adaptability, growth, and sustained success in dynamic environments.

To study the algorithms for exploration and exploitation tradeoff, we create five different bernoulli bandit and the rewards will come from different bernoulli distribution for each bandit. The probabilities assigned to bernoulli bandit are chosen arbitrarily as $[0.005, 0.018, 0.025, 0.029, 0.032]$. In a real world scenario, these probabilities typically are estimated from historical data or other relevant sources to better model the behavior of the bandit.

A/B/n testing

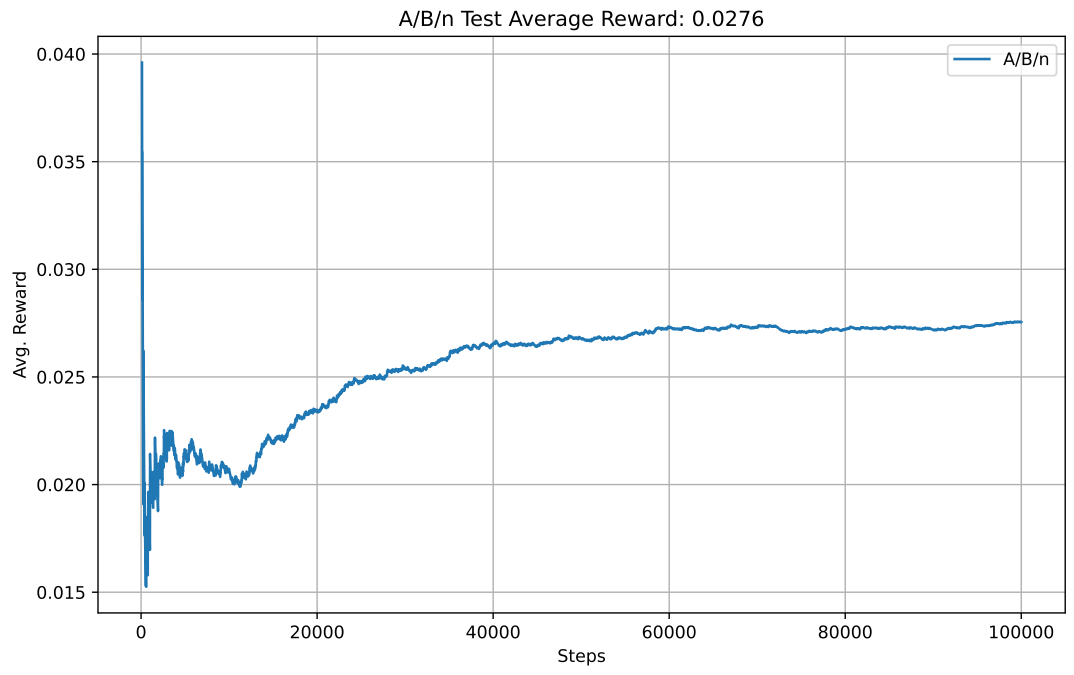
We start with A/B/n testing which is a powerful and widely used methodology in the field of experimentation and data-driven decision-making. This approach involves comparing multiple variations (A, B, and potentially more) of a webpage, application, or marketing strategy to determine which one performs best in terms of a predefined metric, such as user engagement or conversion rate. By randomly assigning users to different variations, A/B/n testing allows for the identification of the most effective design or strategy through statistical analysis. This method provides valuable insights into user preferences and behavior, enabling data-driven optimizations for enhanced performance. While A/B/n testing is a robust tool for experimentation, it requires careful planning and consideration to ensure reliable results, and factors like sample size and test duration are critical to the accuracy of the conclusions drawn from the experiment.We are using the A/B/n testing method as a baseline strategy to compare against more advanced method discussed below.

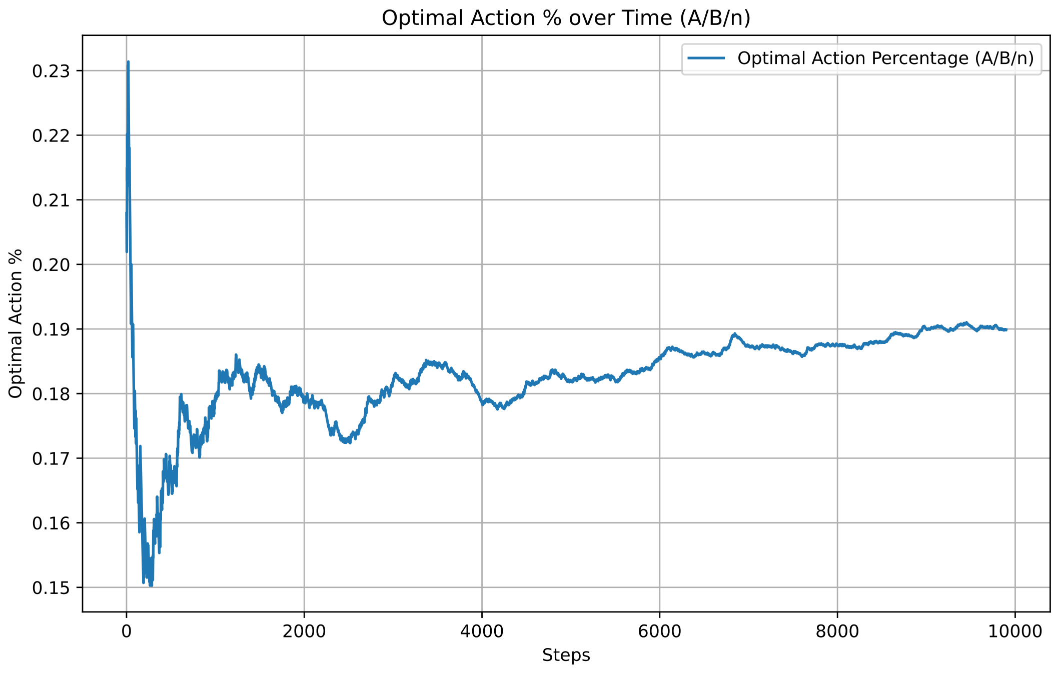
```
## Testing time steps, exploration: 10000

## Production time steps, exploitation: 90000

## Running Algorithm...

## The best performing bandit is D
```





The length of the test period is the hyper-parameter and its choice affects the efficiency of running the test. For nonstationary cases where user behavior changes over time, A/B/n testing has no way of detecting this change.

Suppose we are faced repeatedly with a choice among k different options, or actions. Depending on the action we select, we receive a numerical reward from a stationary probability distribution. The objective to this problem is to maximize the expected total reward over some time period or time steps.

Let us denote the action selected at time step t as A_t and the reward we receive from selecting this action as R_t. The value of arbitrary action a, denoted by $q_*(a)$, is the expected reward given that action a is selected.

$$q_*(a) := E[R_t|A_t = a]$$

If we know the value of each action then we would always select the action with highest value. We assume that we do not know the action values with certainty, although we may have estimates.

Epsilon-Greedy

This method implements the idea that with probability $1 - \epsilon$, choose the action with the highest estimated value (exploit). This leverages the current knowledge to maximize immediate rewards. but once in a while take a random action with probability ϵ (explore).

Let us denote the estimated value of action a at time step t as $Q_t(a)$. The goal is that we want $Q_t(a)$ to be close to $q_*(a)$.

The true value of an action is the mean reward when that action is selected. To estimate this value, one way is to average the rewards actually received. $Q_t(a)$ is calculated as sum of rewards when a taken prior to t over number of times a taken prior to t.

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i I_{A_i=a}}{\sum_{i=1}^{t-1} I_{A_i=a}}$$

where $I_{predicate}$ represents random variable that is 1 if predicate is true and 0 otherwise. By the law of large numbers, as denominator goes to infinity, $Q_t(a)$ converges to $q_*(a)$

The action is selected greedily as:

$$A_t := argmax_a Q_t(a)$$

Incremental Implementation

The averages can be calculated computationally efficient by leveraging incremental method.

Let R_t denote the reward received after the i_{th} selection of this action and let Q_n denote the estimate of its action value after it has been selected $n - 1$ times. Then

$$Q_n := \frac{R_1 + R_2 + ... + R_{n-1}}{n - 1}$$

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

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

The general form of the update rule is:

$$NewEstimate < -OldEstimate + StepSize[Target - OldEstimate]$$

Nonstationary Problem

The method mentioned so far is for stationary problems that is when the reward probabilities do not change over time. Often in the real world, we encounter situations of nonstationarity. In such cases, we give more weight to recent rewards and less weight to past rewards. We can achieve this by using a constant stepsize parameter.

The incremental update rule for updating average Q_n of the $n-1$ past rewards becomes:

$$Q_{n+1} := Q_n + \alpha[R_n - Q_n]$$

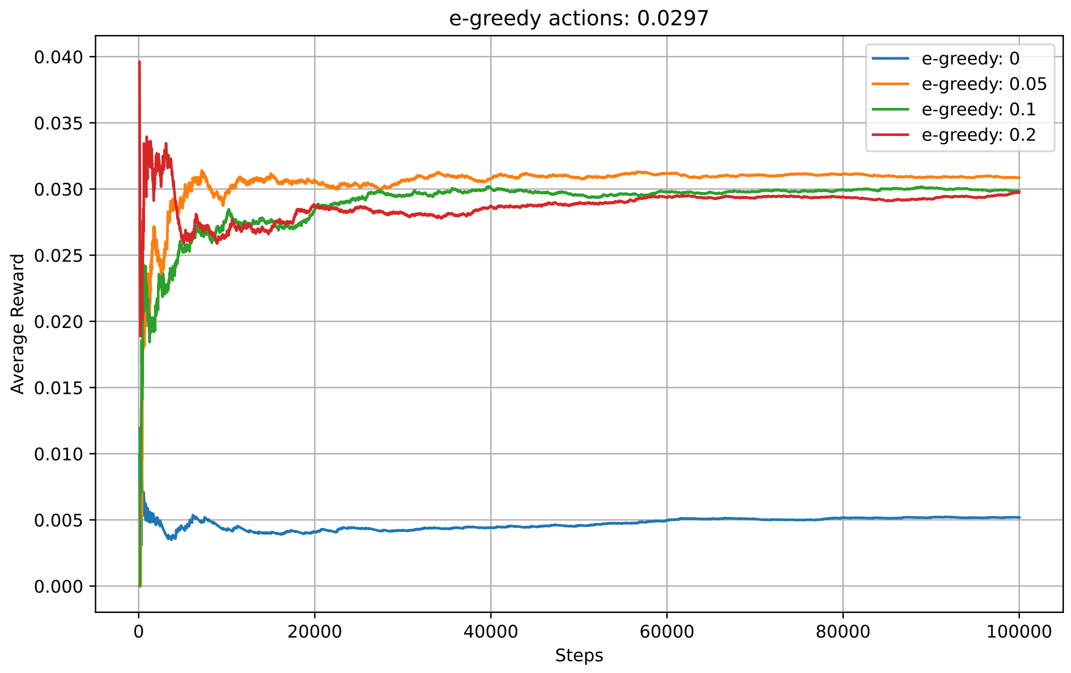
where $\alpha \in (0, 1]$ is constant. This results in Q_{n+1} being a weighted average of past rewards and the initial estimate Q_1:

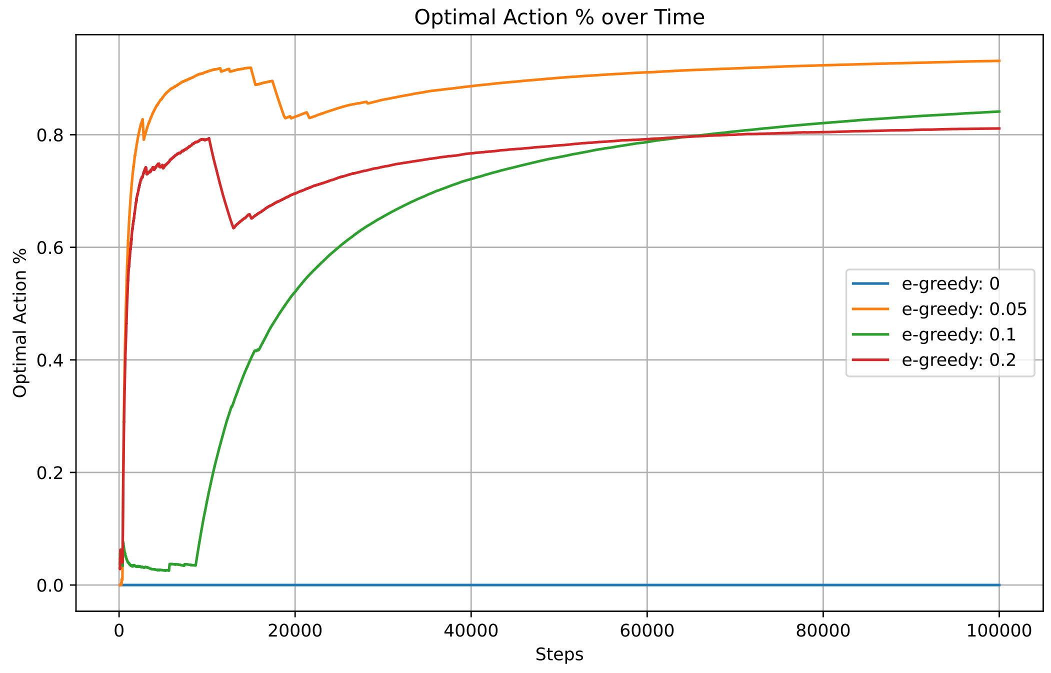
$$\begin{aligned}
Q_{n+1} &= Q_n + \alpha[R_n - Q_n] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)[\alpha R_{n-1} + (1-\alpha)Q_{n-1}] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + ... + (1-\alpha)^{n-1}\alpha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

The performance of epsilon-greedy method can be improved by using optimistic initial action values, that is initialize $Q_1(a) = 3$. This optimism encourages action-value method to explore. Out of all actions, any actions that is selected will have the reward less than the starting estimates, the learner will switch to other actions because it is disappointed with the rewards that it received. All actions will be tried several times before the value estimates converge.

```
## Epsilon Values [0, 0.05, 0.1, 0.2]
## Number of Time Steps 100000

## Running Algorithm...
```





In stationary environments, A/B/n testing and ϵ-greedy approach performs similarly but on non-stationary cases, ϵ-greedy can detect changes and modify the selection which A/B/n testing could not do. As mentioned above, on dynamic environments the update rule becomes $Q_{n+1} := Q_n + \alpha[R_n - Q_n]$.

Upper Confidence Bound

This algorithm operates by assigning each action a confidence bound that incorporates both the estimated value of the action and a measure of the uncertainty associated with that estimation.

At each time step, the UCB algorithm selects the action with the highest upper confidence bound, which is a combination of the estimated value and a term that grows with the uncertainty or lack of exploration of that action. The rationale behind this approach is to encourage exploration by favoring actions with high uncertainty, allowing the algorithm to gather more information about their actual values.

We select the action to take at time t using:

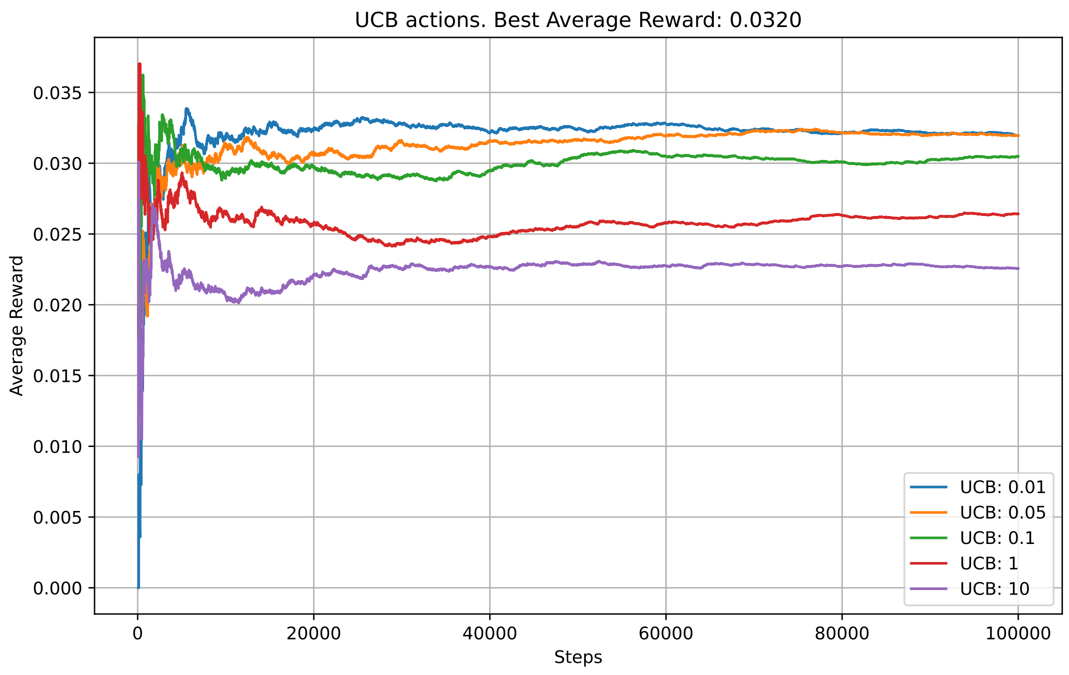
$$A_t := argmax_a \left[Q_t(a) + c\sqrt{\frac{ln(t)}{N_t(a)}} \right]$$

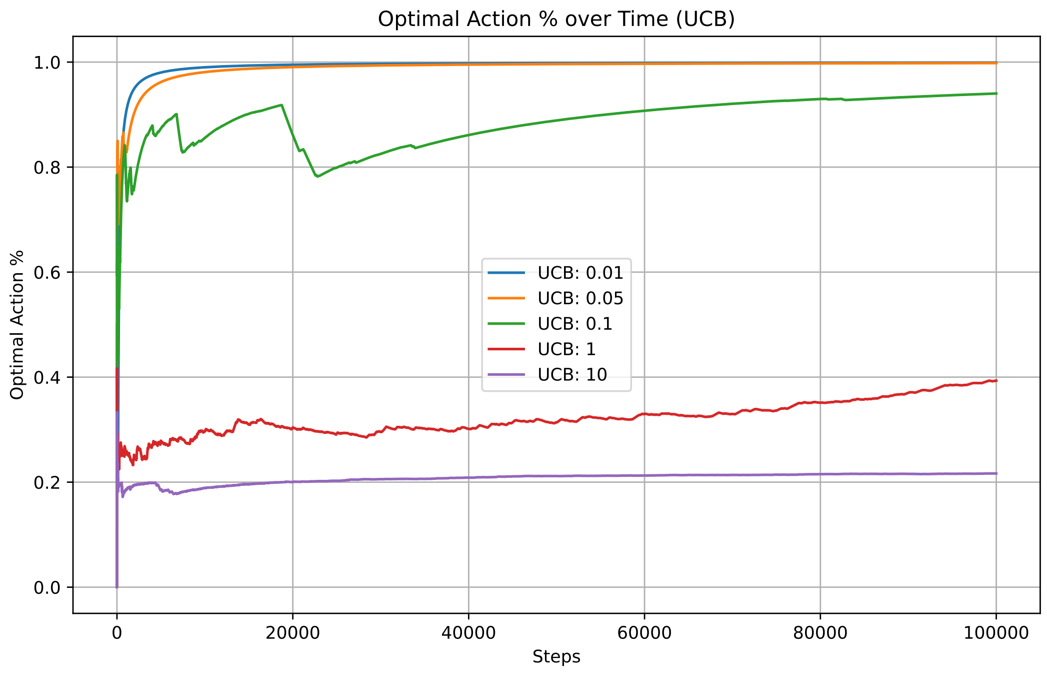
The square root term is the measure of uncertainty for the estimate of the action value of a. The emphasis on uncertainty during decision making is controlled by a hyperparameter c.

```
## Level of Exploration: [0.01, 0.05, 0.1, 1, 10]
```

```
## Number of time steps: 100000
```

```
## Running Algorithm...
```





The UCB algorithm's strength lies in its ability to strike a balance between exploiting actions with high estimated values and exploring actions with uncertain outcomes. As the algorithm accumulates more information through exploration, it tends to converge toward consistently selecting actions with higher true values.

One significant challenge lies in the sensitivity of UCB to parameter tuning, particularly the exploration parameter. Finding the right balance between exploration and exploitation is a non-trivial task and requires careful consideration. Additionally, UCB assumes a stationary environment, meaning it operates under the assumption that the underlying properties of actions do not change over time. In dynamic or non-stationary environments, where action values may fluctuate, UCB might prove suboptimal and slow to adapt. The computational complexity of calculating confidence bounds can also be a concern, especially as the number of actions or the complexity of the problem increases.

Thompson (Posterior) Sampling

Thompson Sampling is a probabilistic approach to solving exploration-exploitation problems. Unlike deterministic algorithms, Thompson Sampling treats the underlying parameters of the problem, such as the true values of different actions, as probabilistic variables. The key idea is to maintain a probability distribution over these parameters and, at each decision point, sample from these distributions to make a decision.

Thompson sampling leverages bayesian inference framework. The bayesian approach treats the parameter as a random variable given the data.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$$

where, $p(\theta)$ is the prior distribution of θ which represents the current hypothesis on its distribution, X represents the data with which we obtain the posterior distribution $p(\theta|X)$. This is our updated hypothesis on the distribution of the parameter given the data we observe. $p(X|\theta)$ is called the likelihood of observing the data X given the parameter and $p(X)$ is called the evidence.

The rules to update for Bernoulli distribution is as follows:

A common choice for the prior is beta distribution in our case. Parameter θ takes values within $[0, 1]$. So, we need to use a probability distribution with the same support to model θ which beta distribution has. In addition, if we use beta distribution for the prior and plug it in bayes formula with Bernoulli likelihood, the posterior also becomes beta distribution. This way, we can use the posterior as the prior for the next update when we observe new data. These are called conjugate distributions and the prior is called ta conjugate prior for the likelihood function. Beta distribution is the conjugate prior for the Bernoulli distribution.

The beta distribution for the prior of k option is given by:

$$p(\theta_k) = \frac{\Gamma(\alpha_k + \beta_k)}{\Gamma(\alpha_k)\Gamma(\beta_k)} \theta_k^{\alpha_k - 1}(1 - \theta_k)^{(\beta_k - 1)}$$

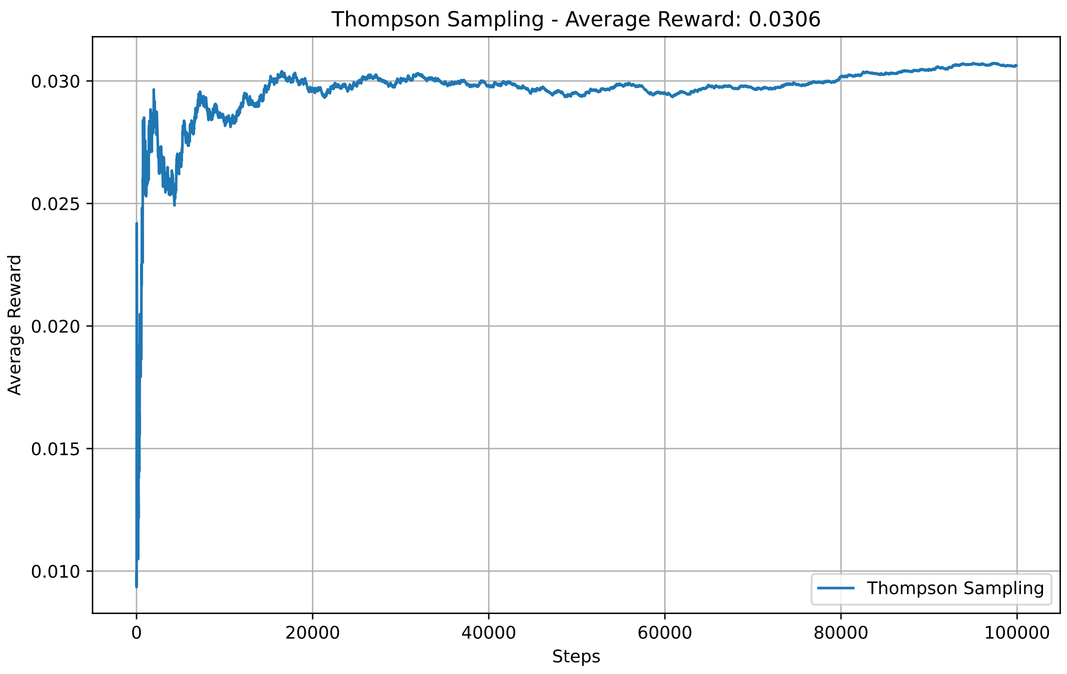
where α_k and β_k are the parameters characterizing beta distribution, $\Gamma(.)$ is the gamma function.

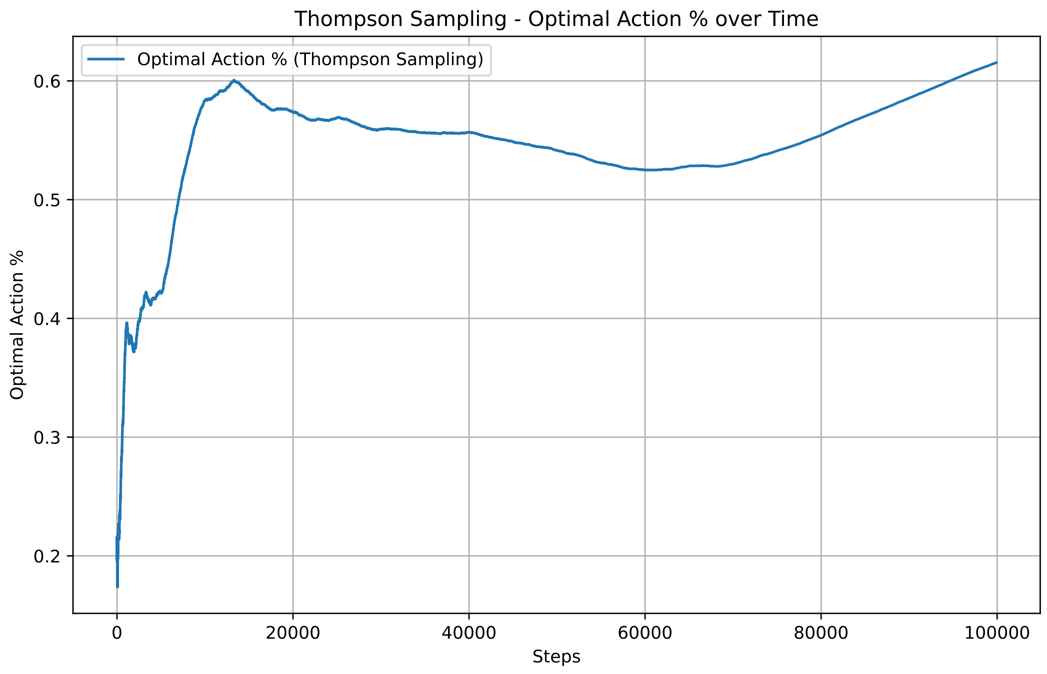
To initialize the prior, we use $\alpha_k = \beta_k = 1$, which makes θ_k uniformly distributed over $[0, 1]$. Once we observe the reward R_t after selecting option k, we obtain the posterior distribution as follows:

$$\alpha_k = \alpha_k + R_t$$
$$\beta_k = \beta_k + 1 - R_t$$

```
## Number of Time Steps: 100000
```

```
## Running Algorithm...
```





Thompson Sampling stands out as a versatile and powerful algorithm for sequential decision-making under uncertainty. Its intrinsic ability to handle probabilistic uncertainty by maintaining probability distributions over action parameters makes it an appealing choice in a variety of applications. Notably, Thompson Sampling adapts well to non-stationary environments, swiftly adjusting its beliefs based on received feedback. The algorithm effectively balances exploration and exploitation, employing a stochastic approach that ensures consideration of less-explored actions while favoring those with higher estimated values. Thompson sampling also did not require us to do any hyperparameter tuning.

Gradient Bandit Algorithms

In this approach, instead of estimating action values and using these estimates to select actions, this method considers learning a numerical preference for each action a. The larger the preference, more often this action is selected. The preference has no intrepretation in terms of reward, only the relative preference of one action over another is important. The action probabilities are determined by $softmax$ distribution as follows:

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

where, $H_t(a) \in R$, $\pi_t(a)$ is the probability of taking action a at time t.

Initially, $H_1(a) = 0$ for all a so that all actions have equal probability of being selected.

On each step, after selecting action A_t and receiving reward R_t, the action preference are updated by:

$$\begin{aligned} H_{t+1}(A_t) &:= H_t(a) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t)) \\ H_{t+1}(a) &:= H_t(a) - \alpha(R_t - \overline{R}_t)\pi_t(a) \quad \text{for all } a \neq A_t \end{aligned}$$

where $\alpha > 0$ is stepsize parameter, $\overline{R}_t \in R$ is the average of reward up to but not including time t with $\overline{R}_1 := R_1$.

The $\overline{R}_t$ is a baseline with which the reward is compared. If the reward is higher than the baseline, then the probability of taking A_t increases.

If you want the comprehensive proof for the updates then please refer to the Sutton and Barto Chapter 2 for details.

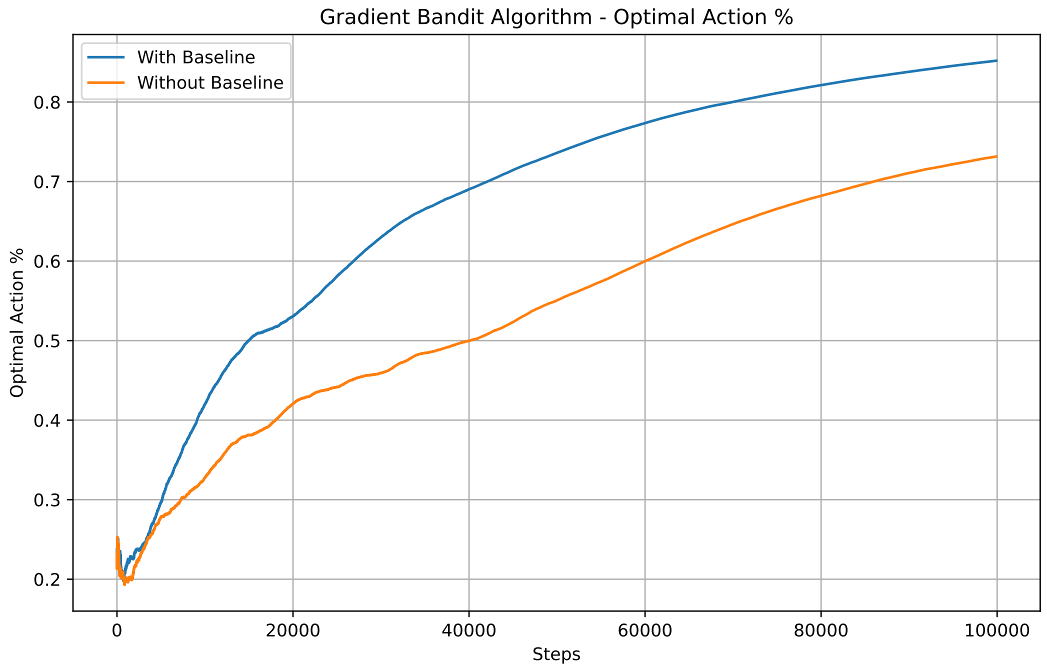
```
## Number of Time Steps: 100000

## The probability of being selected: [0.2 0.2 0.2 0.2 0.2]

## The stepsize parameter: 0.05

## Running Algorithm with baseline...

## Running Algorithm without baseline...
```



Gradient Bandit algorithms offer a distinctive approach to solving the exploration-exploitation dilemma in reinforcement learning settings. One notable advantage is the ability to handle non-stationary environments effectively. By employing a stochastic gradient ascent method to update action preferences, the algorithm can adapt to changing conditions over time. Another strength lies in the simplicity and efficiency, making it suitable for applications with large-scale action spaces. Additionally, Gradient Bandit algorithms naturally extend to problems with continuous action spaces, enhancing the versatility in various domains.

However, the algorithm may be sensitive to hyperparameter choices, particularly the step-size parameter, and the performance can be influenced by the underlying assumptions regarding the reward distributions. Moreover, in scenarios where rewards are sparse or noisy, the reliance on reward values for updating preferences can lead to suboptimal outcomes.