K-Armed Bandit with Non-Stationary Arm Values

CS138 HW1

Joseph Egan

September 2022

Contents

Introduction

The multi-armed bandit is a problem where an agent is given the ability to take **k** actions, where each action can return a positive, negative, or zero value reward to the agent. Over repeated interactions between the agent and the k-arms of the bandit, the agent seeks to learn the optimal action or sequence of actions to take to maximize reward received.

Goal

In a stationary environment, the reward or reward distribution for each action remains constant. This means that once the agent finds the optimal action, that action remains the optimal action forever. For our simulation, we will be comparing how two different methods for estimating the true value of each arm hold up when we are in a non-stationary environment. Non-stationarity is defined by the reward

distribution of each arm changing after every action. Finally, we will propose a potential improvement for action selection and compare both methods after making this change.

Methodology

Setup

We chose to take an object-oriented approach to the creation of this simulation. There is an Agent class that is defined by several variables which toggle key aspects of its policy, such as its method for estimating true arm value and how much it explores. It also has several methods which allow it to perform tasks like initialization, updating value estimates, choosing actions, and resetting itself.

There is also an Action class which defines each of the 10 "arms" of the "bandit". Each one has its own initial value which is generated from a normal distribution with a mean of 0 and a standard deviation of 1. Each instance of the Action class can return a random value from a normal distribution where the mean is equal to its current true value. The arm can then update itself by adding a random value from a normal distribution with mean 0 and standard deviation 0.01. This aspect of the Action class creates the non-stationary environment. Finally, at the end of each cycle, each arm can reset itself.

For our main simulation, we define that there will be 10,000 steps or 10,000 sequential actions taken by the Agent to achieve maximum average reward. These 10,000 steps are repeated 100 times (100 cycles) and the results are averaged.

Initialization

To begin, the Agent will take each action once. Therefore, for steps 0-9, the estimated true value of each arm is irrelevant since there is no decision to be made. For each arm, after the first time it is taken (in initialization) until it is pulled again, the Agent's estimate of that arm's true value is just the first value it received from steps 0-9.

True Value Estimation

The focus of this simulation is to compare how two methods for estimating the true value of each arm compare in a non-stationary environment. The first method we test simply maintains a simple average for each arm of each reward that the Agent received for pulling that arm (taking that action). This can be simplified into the below formula:

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

We call Agent using this model the "Simple Average Method". Where Q(n+1) is the expected true value of an action, R(i) is the reward received from taking that action at the i-th step, and n is the number of times that action has been taken. In a stationary environment, we would expect that after enough times taking each action, all Qs would converge to the true values of each action/arm. However, since those true values are constantly being updated in our simulation, we need to modify this formula to favor more recent rewards over earlier ones. Therefore, the second method we test for estimating the true value of each action is this:

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

Where alpha is a constant. In our code we use alpha = 0.1 and call the Agent using this method "Constant Step-Size Method".

Action Selection

Once we have estimates for the value of each arm, we can then create a policy that dictates which action we take at each step. We will use an epsilon-greedy method that works as follows:

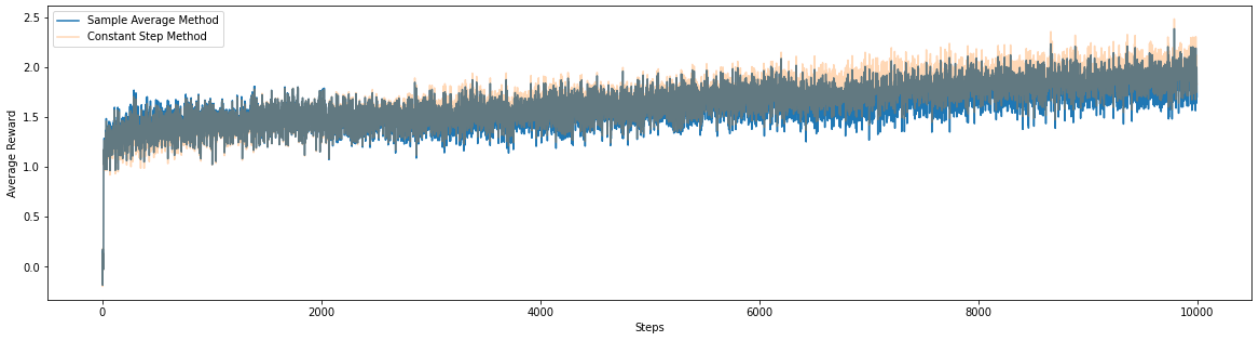
- At each step, with probability (1-epsilon) we will always take the action that has the highest estimated value according to our estimation methods described above. This is referred to as "exploiting".
- With probability epsilon, we will take a random action. This is referred to as "exploring"
- We set epsilon (probability of exploring) at 0.1

The purpose of this method is to maximize reward by taking what the Agent believes to be the optimal action, but to also allow for the possibility that the Agent's value estimates are wrong, and therefore allow it to explore to help correct incorrect estimates. This is especially important given that the true values of each action will be changing at every step.

Analysis of Results

After running the simulation of 10,000 steps 100 times for each method (2 x 100 x 10,000 = 2,000,000 total runs), we achieved the below overall results:

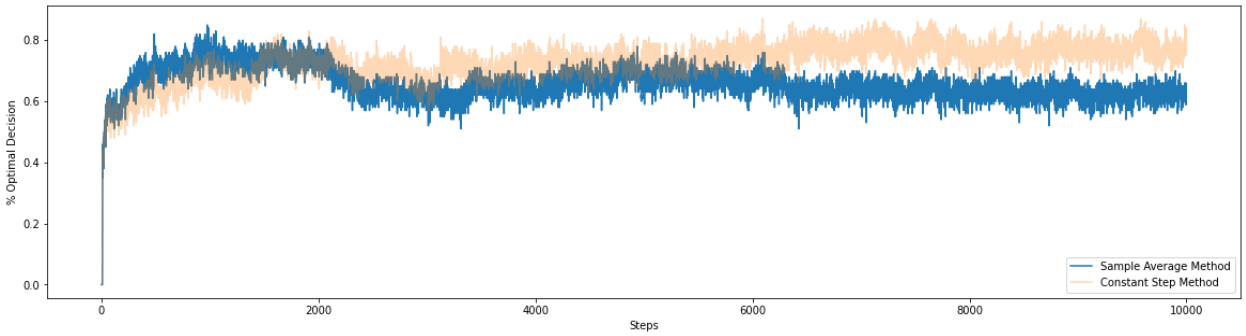
Average Reward Over 10,000 Steps:



The above chart shows the average of the 100 rewards received by both methods at each of the 10,000 steps (10,000 steps taken 100 times each). In this chart we can see that both methods start out roughly equal in ability to estimate true values. However, as the number of steps increases, each action continues to stray further away from its initial and earlier values. As we approach the 10,000th step, the Constant Step-Size Method (which overweighs recent rewards) begins to yield higher average reward.

The supremacy of the Constant Step-Size Method is further suggested when looking at the percentage at each step of each method taking the optimal action, that is the action with the highest true value:

Optimal Action Percentage Over 10,000 Steps:



This above chart shows at each step, what percentage of the 100 times the method took that step, that it chose the action with the highest true value. We can see that over time, the Constant Step-Size Method is better able to capture the changing values of actions, and therefore yields a higher percentage of optimal choices.

Overall Averages:

Model	Average Reward	Average Optimal Action Percentage
Simple Average Method	1.603003	65.7346
Constant Step-Size Method	1.660574	73.0773

Experimentation

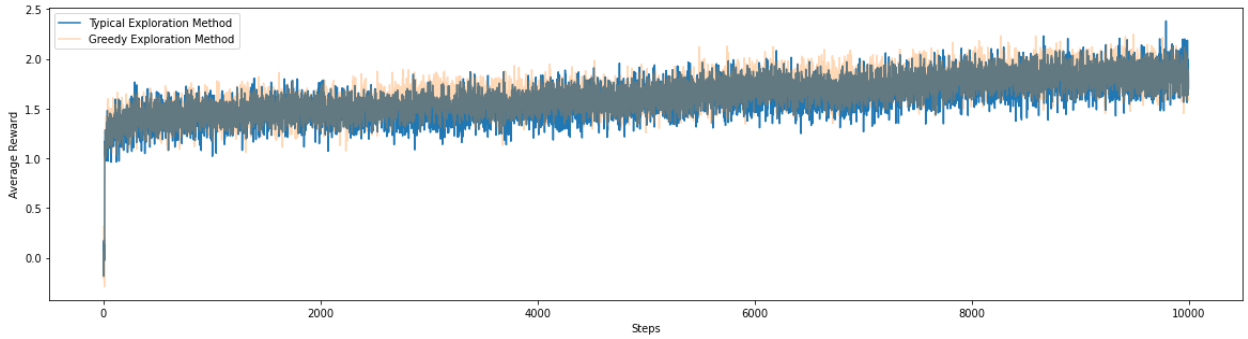
We can take this simulation an additional step by proposing an improvement to the epsilon-greedy action selection policy. Instead of selecting a random action from all 10 possible actions while exploring, our Agents will instead, with probability “c” explore from our top 5 actions with the highest estimated value, and with probability (1-c) select randomly from all possible actions. We call this method “Greedy Exploration”.

Action Selection	Probability
Exploit	1-Epsilon
Greedy Exploration	Epsilon x C
Exploration	Epsilon x (1-C)

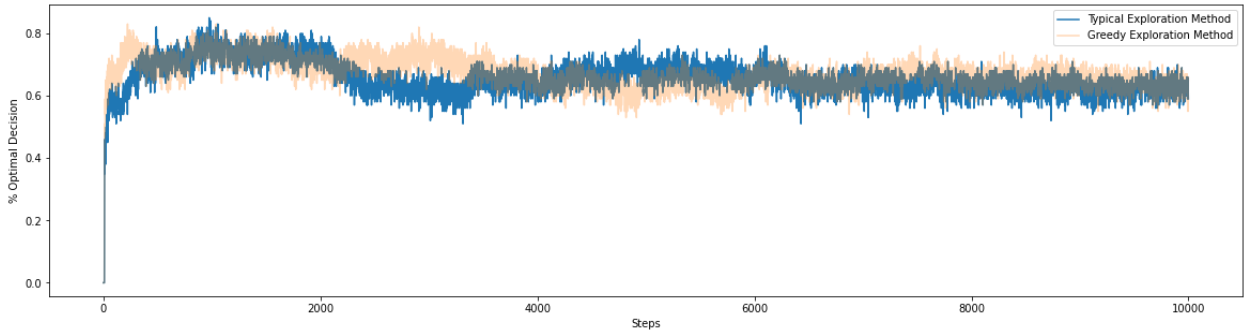
Our hypothesis is that by narrowing the selection of random choices c% of the time, the Agent will select the optimal choice more frequently, since that action has a higher true value (distribution mean) and therefore sampling from that distribution will more often yield a higher reward. We set “c” at 0.5.

First, we test this on the Simple Average Method Agent:

Average Reward Over 10,000 Steps:



Optimal Action Percentage Over 10,000 Steps:

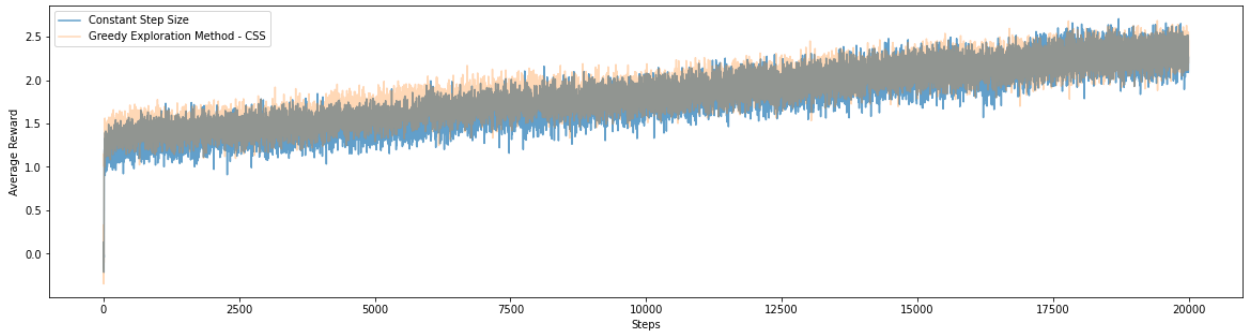


At first glance, the differences between the two appear to be insignificant. However, when looking at overall averages, it appears that this modification has slightly improved the decision making of the Agent:

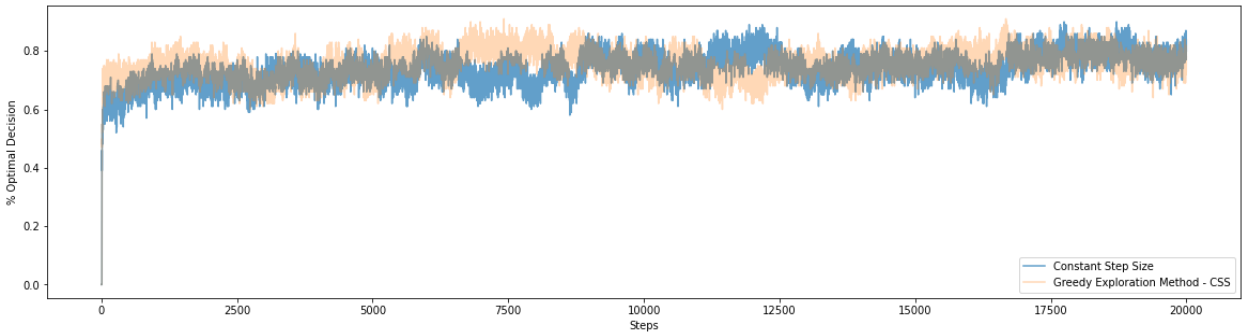
Model	Average Reward	Average Optimal Action Percentage
Simple Average Method	1.603003	65.7346
Simple Greedy Exploration	1.652320	67.2424

We next apply this modification to our Constant Step-Size Method Agent and increase the number of steps per cycle to 20,000 to see if the differences become more visible in the charts:

Average Reward Over 20,000 Steps:



Optimal Action Percentage Over 20,000 Steps:



Again, the differences appear to be insignificant from the charts, but looking at overall averages, it is clear that this modification has improved the Agent's decision making:

Model	Average Reward	Average Optimal Action Percentage
Constant Step-Size Method	1.79695	74.03120
Constant Step-Size Greedy Exploration	1.87277	75.76335

Conclusions

Based on our results, we determined that while neither method can perfectly estimate the true values of non-stationary actions values, using a constant step size is more accurate than a simple average. Furthermore, by increasing the frequency by which the Agent explores the actions with the highest estimated values, we are able to increase the Agent's average reward and percentage of optimal actions over 10,000 and 20,000 steps.

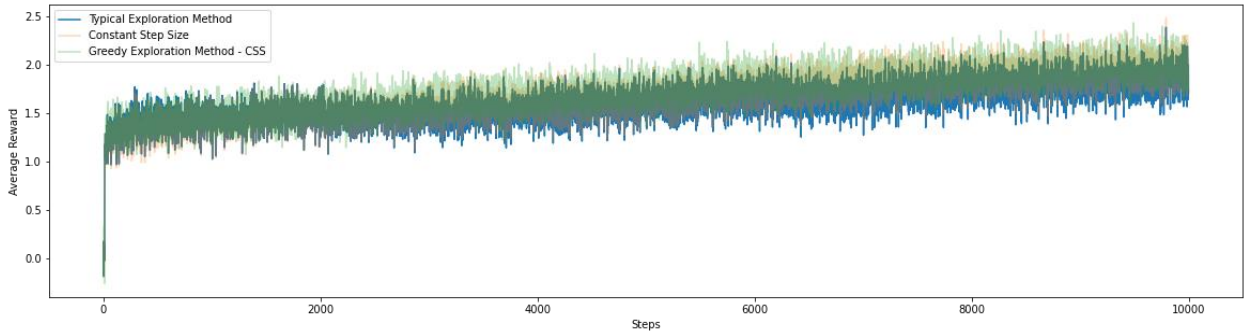
Future Improvements/Questions

Given that we have several numerical variables that dictate the nature of the simulation, it is only natural that we suggest that future simulations and experiments could adjust values like alpha, epsilon, number of steps, and our experimental variable, "c". An additional loop could be created (on top of the 100 cycles per method), that loops through several values for each variable while running the simulation. We would then be able to make similar comparisons as we did in this problem, but across several methods/variable values.

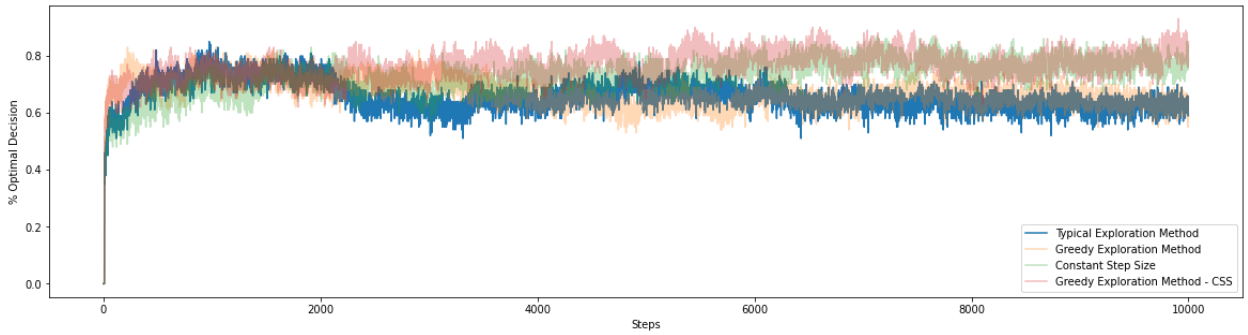
Additionally, we could sample different value estimation methods that are given in the textbook and compare how those methods stand up to our simple average and constant step-size.

Additional Charts and Tables

Average Reward of Simple Average, CSS, and GECSS methods over 10,000 steps:



Optimal Action Percentage Over 10,000 Steps, all models:



Final Averages with Final Agents:

Model	Average Reward	Average Optimal Action Percentage	Steps
Simple Average Method	1.603003	65.73460	10,000
Constant Step-Size Method	1.796950	74.03120	20,000
Simple Greedy Exploration	1.652320	67.24240	10,000
Constant Step-Size Greedy Exploration	1.872770	75.76335	20,000

Bonus Questions

Exercise 2.8: UCB Spikes In Figure 2.4 the UCB algorithm shows a distinct spike in performance on the 11th step. Why is this? Note that for your answer to be fully satisfactory it must explain both why the reward increases on the 11th step and why it decreases on the subsequent steps. Hint: If c = 1, then the spike is less prominent.

Answer:

The UCB algorithm starts by selecting each of the 10 actions once due to the fact that when Nt(a) = 0, the algorithm values the action “a” infinitely higher than any action already taken. It’s likely that on the 11^{th} step, the algorithm will choose the action that gave it the highest reward from the first 10 steps, and it is likely that given that the first reward of that action was high, that it came from a distribution with a higher mean than the others. Therefore, it is likely that the reward will be high again, causing the spike. After the 11th step, the algorithm will favor an action that has been selected fewer times, which

will likely lead to taking an action with a reward distribution with a lower mean, causing the spike to go back down.

Exercise 2.10 Suppose you face a 2-armed bandit task whose true action values change randomly from time step to time step. Specifically, suppose that, for any time step, the true values of actions 1 and 2 are respectively 10 and 20 with probability 0.5 (case A), and 90 and 80 with probability 0.5 (case B). If you are not able to tell which case you face at any step, what is the best expected reward you can achieve and how should you behave to achieve it? Now suppose that on each step you are told whether you are facing case A or case B (although you still don't know the true action values). This is an associative search task. What is the best expected reward you can achieve in this task, and how should you behave to achieve it?

Answer:

If we do not know which case we are in, then the highest expected reward we can achieve is 50. We can achieve this by picking 1 every time .5(10) +.5(90) or picking 2 every time .5(20) + .5(80). Also, we could achieve this by just picking randomly every time .25(10) + .25(20) + .25(90) + .25(80). However, if we know which case we are in, we can pick optimally knowing that for case A, action 2 yields higher reward, and for case B, action 1 yields a higher reward. In this dynamic, we are able to earn an expected average reward of .5(20) + .5(9) which is 55.