Improved Upper Confidence Bound Algorithm Using Adaptive Exploration Parameter for Non-stationary Environment

Jarry Zhu
Department of Computer Science
McGill University
Montréal, H3A 0G4
zeyuan.zhu@mail.mcgill.ca

Beiyu Li
Department of Computer Science
McGill University
Montréal, H3A 0G4
beiyu.li@mail.mcgill.ca

Abstract

In this project, we try to improve the performance of Upper Confidence Bound (UCB) algorithm in non-stationary environment by using an adaptive exploration parameter. We then compare its performance with Epsilon Greedy and traditional UCB algorithms under same environment using the same sets of environment parameters. Jarry came up with the idea of this project, set up the appropriate environment for testing, and run the experiments. Beiyu helped running the experiments, analyzed the plots with Jarry and finalized the write-up.

1 Introduction & Related work

The multi-armed bandit problem is a classic scenario in reinforcement learning, used to describe decision-making situations where a choice must be made between multiple options to maximize cumulative reward over time. There are various strategies that can be employed to solve this kind of problems, for example, epsilon-greedy algorithms, Upper Confidence Bound (UCB) algorithm and Thompson Samplings.

In assignment 1, we tested the performance of these strategies in both stationary and non-stationary environments. We found that UCB algorithm had some ability to handle a changing environment. In our project, we aim to design a variant of the UCB algorithm to better handle environments where the reward distributions change over time, which requires modifying the traditional UCB to dynamically adjust its exploration-exploitation balance based on the detected changes in the environment.

Lots of effort has been put into modifying the UCB policy to make it more suitable for non-stationary environment. Discounted UCB was introduced by Kocsis and Szepesvári(2006) **(author?)** [3] , which involves using a discount factor to rewards and observation counts to give more recent observations more weight. Another algorithm, sliding window UCB (SW-UCB) was proposed by Garivier and Moulines (2008)**(author?)** [2], which uses a sliding window to include only the last certain number of observations for each arm when calculating the mean and the number of times arms are pulled. Cheung et al. (2019)**(author?)** [1], developed a novel Bandit-over-Bandit (BOB) algorithm that tunes SW-UCB adeptly to deal with cases where the upper bound of total change are not available.

In this project, we want to find out whether a much more simpler improvement, by adjusting the exploration parameter c in UCB can get more improved performance in the non-stationary environments. We will compare our version of UCB with traditional UCB, epsilon-greedy algorithms in different environments, with various scale and frequency of changes in reward distribution.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2 Methodology & Experiment Set-Up

2.1 Experiment Set-up

2.1.1 Bandit Set-up

We configure a bandit with a specified number of arms, each arm generating rewards that follow a normal distribution with dynamically changing means. The setup involves initializing the bandit with specific parameters including the number of arms, their initial means, a variance multiplier that determines the reward distribution's variability, a probability of mean change that determines how frequently the mean values are altered, and a change factor that controls the magnitude of each mean change.

The step function changes the means of the arms based on the change probability according to a Gaussian noise addition, reshuffling, and clipping to make sure values fall within a practical range (-10 to 10) in case a very extreme arm is generated.

Each simulation step involves pulling one of the arms, calculating the reward based on the current mean and a derived variance, and calling the step function to potentially update the arms' mean.

For our project, we used 5 different 5-armed bandits with initial means of 0.1, 0.3, 0.5, 0.7, and 0.9 respectively for each arm. Except for the one that doesn't change its mean, for the parameters of the rest of the bandits, we used the following configurations:

(0: stationary) bandit that has a fixed mean of initial mean

(1: Baseline) variance multiplier=1, change factor =1, probability of mean change=0.005; this configuration has an average reward variance of 0.01 * variance multiplier * arm means, when its reward changes the mean alter values' variances are arm mean * 1 * variance multiplier, and the final change value is *= change factor

(2) variance multiplier=5, change factor =1, probability of mean change=0.005; The variance of rewards and mean alter value is 5 times higher than the baseline

(3) variance multiplier=1, change factor =5, probability of mean change=0.005; The scale of mean alter value is 5 times higher than the baseline

(4) variance multiplier=1, change factor =1, probability of mean change=0.025. The mean is 5 times more likely to be altered frequency-wise

This setup allowed us to explore the performance of each algorithm in a non-stationary environment with different extents and frequencies of changes.

2.1.2 Improved UCB Algorithm

In our improved UCB algorithm, we have the same parameters as the traditional UCB algorithm, and an extra "c update scale " parameter to determine how much the exploration parameter c will be updated and an extra reset boolean variable.

The selection of arm pulled followed the same rule as the traditional UCB algorithm.

$$\mathrm{A}_t = \arg\max_a \left(Q_t(a) + c \cdot \sqrt{\frac{\log t}{N_t(a)}} \right).$$

To update the exploration parameter c, we calculated not only the upper bound but also the lower bound, forming a confidence interval (CI).

$$\mathrm{CI}_a(t) = \left[Q_t(a) - c \cdot \sqrt{\frac{\log t}{N_t(a)}}, Q_t(a) + c \cdot \sqrt{\frac{\log t}{N_t(a)}} \right]$$

If the current reward falls within the CI, we will update c using the following formula: c = c * (1 - c update scale), which decreases the c to reduce exploration. However, if the current reward lands outside of the CI, if reset is true, c will return to its default value specified in the model parameter(c change only causes more exploitation when applicable). Otherwise, we will update c by another formula: c = c * (1 + c update scale) to allow more exploration as well as exploitation, so the c is free to move to any value.

2.1.3 Algorithm Parameters

For the epsilon-greedy algorithm, we used the epsilon values of 0.25, 0.125, 0.0625, alpha values of 0.05, 0.2, 0.1 and averaging.

For traditional UCB, we used alpha values of 0.05, 0.2, 0.1, and averaging, and c values of 0.125, 0.5, 1, 2.

For improved UCB, we used the same alpha and c values as traditional UCB. We set the c alter values to be 0.5, 0.25, 0.125, 0.0625, 0.03125 and 0, and ran experiments with reset set to true and false respectively.

2.2 Methodology

For each configuration of each algorithm, we run 100 independent runs, each consisting of 1000 time steps. We plotted the following graphs:

(1) The instantaneous reward at each time step averaged over the 100 runs;

(2) The reward received over time, averaged at each time step over the 100 independent runs;

(3) The fraction of runs(out of 100) in which the true best action (the action with mean > 0.999*the highest mean value)is estimated best; (This is the most important dimension due to the randomness of the normal result of arms)

(4) The total regret up to time step t averaged over the 100 runs

For the improved UCB, we also plotted a graph showing the value of c at each time step averaged over the 100 runs for evaluation.

3 Experiment Results

3.1 Comparison of Algorithms

The best configuration for UCB, epsilon greedy, and our improved UCB were obtained for each bandit environment, and in general, our improved algorithm worked much better in a highly dynamic environment with the cost of negligible extra O(1) space and extra O(T) calculation to get lower confidence bound. However, our UCB may have a slightly worse performance than the traditional UCB if c is reset to input instead of moving freely within the episode and the environment tends to be stable

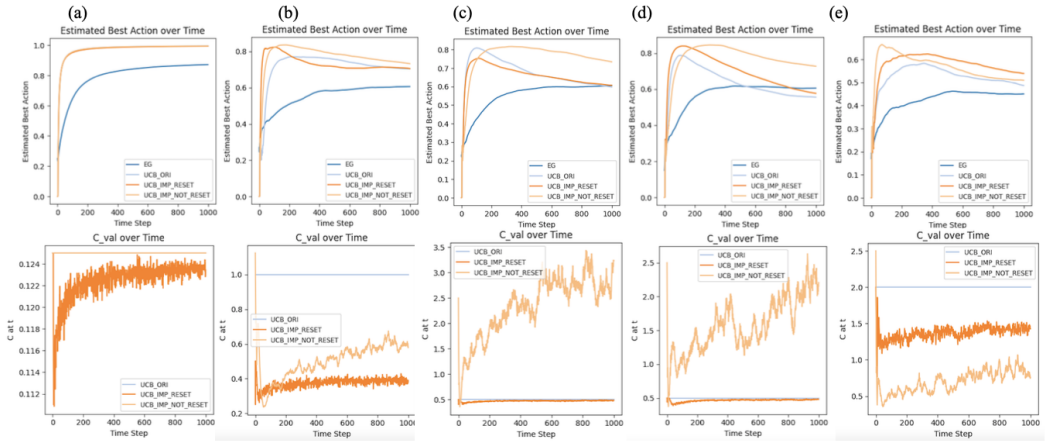

Figure 1: Performance and c values of different algorithm in different environments. (a)-(e): Bandit environment 0-4.

3.2 Effect of original UCB parameters on our algorithm

It is clear that as we used a product-based way to adjust c, it was able to move quickly to a desired range with a higher or lower baseline c from 0.125 to 2, and alpha is generally not so significant as a high c alter value would dominant how the bounds moves.

3.3 Effect of c Alter Factor

We have tested how different c alter factors affect the performance of the algorithm, using good traditional UCB setups as baselines.

It seems that if we increase c when we detect a change in environment, a reasonably larger c alter factor (0.25 even 0.5) would lead to a better performance as it is able to move faster towards the appropriate value and in a few pull it will be able to detect the best arm quickly enter and remain in exploit mode (appropriately low c) as long as possible.

c alter factor can also control how close it can reach the true preferred c range as c value can be dragged by relatively smaller c alter factor.

This is further supported as the our algorithm can achieve much better (same config used) result if it "reset" is false than true when the preferred C is above the original value, if the c prefered is under original c then the performance difference is much smaller and we assume the gap remained was because the c changing too much and it always take more time to stabilize.

It is interesting that the Best c our algorithms wonders around shows up in our traditional c test but doesn't serve as an optimal value, seems that a constant c would have great trouble handling a environment changes quickly.

3.4 Effect of "Reset"

Our further comparison even shows that "Reset" may only be usable if the environment is highly non-stationary ($p = 0.025$), for most environments it performs worse than certain fixed c, usually with a smaller alpha so a longer temporal exploit is allowed (often better than many other fixed c) and we believe that is because in many cases the optimal performance of our algorithm is obtained because our c, base on needs can move quickly/ stay stable in the preferred range and the "reset" of c to input is only appropriate when such rapid move is suitable (highly unstable environment) and the preferred c is below the baseline c input.

4 Conclusions & future work

4.1 Conclusion

It seems that our UCB variant greatly improved its ability to handle environments changes frequently while maintain a decent performance against static environment. Different c alter factors can control how close and fast for our algorithm reach and stay in its preferred c range for the environment, and the "reset" mode is only useful if preferred c is below the baseline c and the environment is very unstable

4.2 Future work

As Upper CI calculation takes most of UCBs' computation complexity. We wonder, as our algorithm's CI is calculated at each time step, it might worth a try to save computation by calculating CI in longer interval.

Other evaluation dimension would also help greatly for such type of environments.

Combining this idea with other UCB variants, such as SW-UCB might also worth a try but it might take great effort to combine them and we believe that sort of method needs a much more complicated testing environment

References

References

[1] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Learning to optimize under non-stationarity, 2021.

[2] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems, 2008.

[3] Kocsis and Szepesvári. `https://www.lri.fr/~sebag/Slides/Venice/Kocsis.pdf`, 2006.