COMP 138 RL: Programming Assignment 1

Jane Slagle

September 28, 2024

1 Introduction

1.1 Problem Background

This programming assignment is centered around k-armed bandit problems, specifically focusing on a testbed with 10 arms. In the context of k-armed bandits, an agent repeatedly selects one action from a total of k different arms, each referred to as an action. Each time the agent selects an action, it receives a payout in the form of a numerical reward value. A 10-armed testbed means that the agent will be interacting with an environment that has 10 different actions to choose from.

Bandit problems can be categorized as either stationary or non-stationary. Here, we will focus on a non-stationary environment, meaning that the distribution changes over time.

1.2 Relation of Problem to Reinforcement Learning

k-armed bandit problems differ from traditional reinforcement learning problems in that they do not include a notion of state. Instead, k-armed bandit problems focus on the actions taken, which allows them to achieve their main goal of maximizing reward over time. This setup enables us to concentrate on the trade-off between exploration and exploitation when running through k-armed bandit problems. Exploration entails selecting a random action with probability epsilon, a value that is predetermined, while exploitation calls for choosing the action that most maximizes the value function.

Some specific methods for determining the best agent are outlined in the following sections, where I describe a 10-armed testbed that I explored in Python with a non-stationary approach.

2 Problem

2.1 Problem Statement

Design and conduct an experiment to demonstrate the difficulties that sample-average methods have for nonstationary problems. Use a modified version of the

10-armed bandit testbed in which all the $q_*(a)$ start out equal and then take independent random walks (say, by adding a normally distributed increment with mean 0 and standard deviation 0.01 to all the $q_*(a)$ at each step). Prepare plots like Figure 2.2 for an action-value method using sample averages, incrementally computed, and another action-value method using a constant step-size parameter, $\alpha = 0.1$. Use $\epsilon = 0.1$ and longer runs, say of 10,000 steps.

2.2 Approach

In this assignment, we will implement a 10-armed bandit, where each arm represents a different action that the agent can take in the environment at each time step. Each arm starts out with a same initial reward value which we initialize from a normal distribution with mean 0 and variance 1, which supports our motivation to choose initial reward values to be so. This corresponds with the 10-armed bandit method described in the book [1] which states that the initial action values are chosen according to a normal distrubtion with mean 0 and variance 1. Then, at each time step, increments itself by a random normal distribution with mean 0 and standard deviation 0.01. Recall from Section 1.1 that we are taking a non-stationary approach. This implies that the reward probability distribution of each action changes over time, specifically by adding a normal distribution centered around the reward value, with randomness added. For the randomness, we will use a variance of 1 and for the mean we will use the reward value for the selected action. This follows the method described in Section 2.3 of the book [1].

The role of the agent here is to interact with the 10-armed testbed environment that we are simulating by selecting the action to take and updating its action-value estimates based on the rewards computed at each time step.

In our implementation, we will employ two methods for updating these action-value estimates. The first being an action-value method using incrementally computed sample averages as described in Section 2.4 of the book [1]. The second being an action-value method with a constant step size parameter, alpha = 0.1. The results of updating the action-value estimates via these two methods will be discussed in the Results Section of this paper.

We will also implement two different approaches for action selection. The first is the approach outlined in the problem statement, an epsilon-greedy action selection method with epsilon = 0.1. And then, for an additional question to explore here, we will implement an UCB (Upper Confidence Bound) action selection method. The performance of these action-selection strategies over our two action-value updating methods will be discussed in the Results Section of this paper as well.

2.2.1 Action-Value Methods

As discussed above, we consider two different action-value methods in our implementation, sample averages incrementally computed and using a constant step size parameter.

These methods are described in Section 2.4 of the book [1], and from that description, we know that both methods update the action-value estimates with equation 2.3:

$$Q_{n+1} = Q_n + step[R_n - Q_n]$$

where *step* represents the step size corresponding to the method we are using. Q_n stores the action-value estimate of the action on the current time step while R_n stores the reward of that action. The two methods differ only in terms of the step size used in the above equation:

1. With the sample averages incrementally computed approach, the step size used in the equation is $1/n$, where n is the number of times the action has been selected.
2. In the constant step-size approach, the step size is a fixed value, given by alpha $= 0.1$ in this case. It does not take into account the number of times the action has been selected.

2.2.2 Action-Selection Methods

As discussed above, we consider two different action-selection methods in our implementation, epsilon-greedy action selection and UCB (Upper Confidence Bound) action selection.

In our implementation we are using epsilon-greedy action-selection with epsilon $= 0.1$. This means that with a probability of epsilon, 0.1, that when the agent selects which action to take next, it picks a random action with probability epsilon (exploration) and then all other times, it selects the action with a greedy approach, meaning that it selects whichever action has the current highest estimate value (exploitation) with a uniformly random distribution.

We also analyzed the problem with UCB action-selection. From equation 2.10 in the book in Section 2.7 [1], we know that with this approach the agent selects actions according to the following formula:

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

where c is a constant nonzero positive hyperparameter value that is predetermined that controls the degree of exploration and $N_t(a)$ is the number of times that action a has been selected prior to the current time step t. If $N_t(a)$ is equal to 0 then action a is considered to be the maximum action.

2.2.3 Additional Question Asked

For the additional question portion of this assignment, I decided to use an alternative approach to the epsilon-greedy action selection specified in the problem statement and also analyze the problem using UCB (Upper Confidence Bound) action-selection. Since we were already analyzing the problem with two different action-value methods, I thought it would be interesting to see how those two action-value methods compared across two different ways of doing action selection. I was also inspired by Section 2.7 of the book [1] which explains how UCB is an alternative to the epsilon-greedy approach. Section 2.7 states that the epsilon-greedy might now always be the best method to use because it only looks at the optimal actions at the current time step, so UCB could be an improvement over epsilon-greedy as it selects among the non-greedy actions according to their potential for being optimal. The results of analyzing this additional question are given below in the Results section.

2.3 Motivation Behind Approach

Running this experiment with the above described approach will allow us to gain insight into the negatives associated with solving non-stationary problems with reinforcement learning methods.

By using sample-average methods for updating the action-value estimates in a non-stationary environment, we will hopefully learn and understand why sample-average methods perform poorly when the reward distribution is changing over time. This will then provide us with background knowledge with future reinforcement learning tasks when we are deciding how best to design an experiment.

By comparing the two action-value updating methods described above, we will learn more about how the choice of learning rate (constant vs. non-constant) impacts the agent's ability to learn in the environment. This comparison will highlight the effects of placing importance on more recent rewards versus placing the same importance on all rewards.

By examining the different action-selection approaches, we will gain a better intuition on how best to balance the exploration-exploitation trade-off that occurs with reinforcement learning tasks.

3 Experiment

3.1 Code

In my code implementation, I wrote three files, *testbed.py*, *agent.py*, and *main.py*.

I wrote *testbed.py* to represent the non-stationary 10-armed testbed environment that we are running the environment in. I wrote *agent.py* to represent how the agent interacts with the environment and *main.py* then actually runs the experiment on 2,000 runs and 10,000 time steps. It runs the environment with the sample average and constant size methods using both epsilon-greedy action selection and UCB action selections, creating plots for each action-selection method that show the average reward over the 10,000 time steps and the percentage of optimal actions chosen over the 10,000 time steps for both action-value methods.

In terms of parameters, epsilon = 0.1, alpha = 0.1 and c = 2 were used throughout the code. I decided to use c = 2 for UCB action-selection as that was the value used based off of Figure 2.4 in the book [1], it used c = 2 on a 10-armed testbed, a similar problem to what we are implementing here. I also thought c = 2 would be a good moderate choice for both exploration and exploitation since c controls the degree of exploration, so a higher c means more exploring and a lower c means more exploitation, so c = 2 seemed like a neutral choice for both.

4 Results

4.1 Results Expectations

For the plot of Average Reward, I expect the constant alpha step size method to perform better than the sample average method over the 10,000 time steps that the experiment will be run over. With the constant step size method, the reward values will be updated with a fixed step size of alpha = 0.1. Comparing this to the sample average method, the reward values will be updated based on the average of all of the previous rewards received for that action and uses a step size of the number of times the action has been chosen. This means that as time goes on in the 10,000 time steps and as the number of times that an action was selected increases that the step size will decrease. This in turn means that rewards from older previous time steps will have more weight to them than more previous time steps, so the newer previous rewards will contribute less to the average. This suggests that the sample average method results will be slower to change and adapt to the environment as it too changes.

For the plot of Percentage Optimal Action, I expect to see similar results of the constant step size method performing better than the sample average method. This is because of the exact same reasoning of the sample average method not estimating as good as results since it is slower to adapt to changes in the environment. Thus, I believe that the constant step size method will be able to reach a higher percentage of optimal actions over the 10,000 time steps than the sample average method.

I would expect to see these same results for both the epsilon-greedy and UCB action selection methods.

I would expect the epsilon-greedy results to perform better than UCB in this scenario. In Section 2.7 of the book [1], the authors state that UCB methods have difficultly dealing with non-stationary problems, and in this problem, we are implementing the environment as a non-stationary one. Thus, I think that the epsilon-greedy approach will be better suited for the problem at hand here.

4.2 Original Experiment Results

The results of the original problem statement are given in the below plot. The plot shows the results from running the experiment with epsilon-greedy action selection.

The results of Figure 1 are mostly consistent with the expectations described

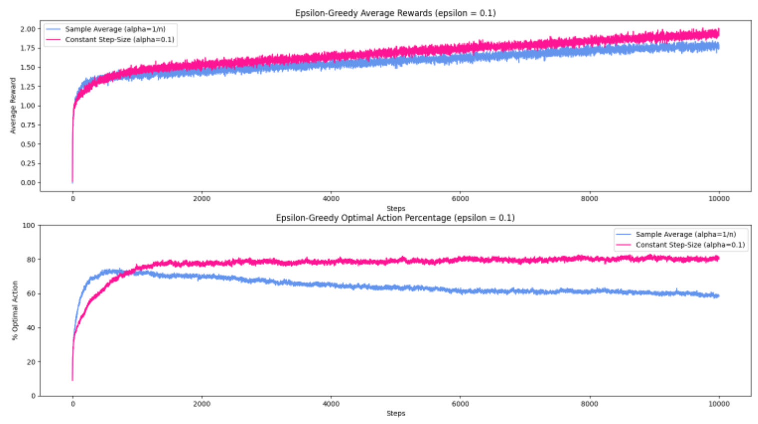

Figure 1: Epsilon-greedy results of average rewards for sample average and constant step size methods over 2,000 runs and 10,000 time steps. The above plot shows the average rewards, and the below plot shows the optimal action percentage of the two action-value methods.

in Section 4.1.

We see in both subplots, for both average reward and percentage optimal action that the constant step size action value method overall performs better over the 10,000 time steps than the sample average method. This makes sense as the sample average method updates the action-value estimates based on all of the previous rewards received for a given action. As the time steps increase up to 10,000, rewards from the time steps earlier on have higher weights to them in the sample average method which means that they influence the action-value estimate too much which gives results that are not as accurate. This is exactly what we observe in Figure 1 with the average reward of the constant method climbing to about 2.0 compared to about 1.7 for the sample average over the 10,000 time steps and with the constant step size method reaching about 80 percent of optimal action percentage with the sample average method reaching only about 60 percent over the 10,000 time steps.

Looking at the plot of Average Reward from Figure 1, we note that both the sample average and constant step size methods start out with an average reward value of 0. This is expected since at time step 0, no actions have been taken yet so there are no rewards for which to update the action-value estimates with.

We also observe that the sample average average rewards are initially better than the constant step size method until about 200 time steps. This is also expected as in initial time steps, the sample average method is able to perform better since it uses all of the past data up until that point to make decisions with. When the number of time steps is low, there is not a lot of data yet so this yields more accurate results. However, from Figure 1, we see that after about 200 time steps, the constant step size method then gradually and consistently performs better and better as the amount of time steps increases. This is also expected as we ran the experiment in a non-stationary environment which means that more recent rewards and action-value estimates have more importance placed on them when making decisions. The constant step size has an advantage over sample average here as at higher time steps, it is able to give more weight to those recent values (since it weights all rewards the same).

Looking at the optimal percentage action plot from Figure 1, we observe the same features of both methods starting out at 0 and the sample average initially performing better. These characteristics of the plot can be justified in the same way as they were for the average reward plot. Over the 10,000 time steps, the constant step size reaches and levels off at around 80 percent. Comparing this to the sample average results which do not level off as clearly as the constant step size method, but they do stay consistently around 60 percent for the majority of the total time steps. This can be explained by the differences in how the two methods act in the non-stationary environment. The constant step size takes longer than the sample average method to explore, it takes the agent longer to learn in the environment and to adjust the action-value estimates based on the rewards. This will lead to the constant step size parameter being able to reach a higher optimal action percentage, which is what we observe in Figure 1. Overall, both methods stay consistent at their respective percentages without any significant changes over the total time steps.

4.3 Additional Question Asked Results

The results of the additional question asked are given in the below plot. The plot shows the results from running the experiment with UCB action selection.

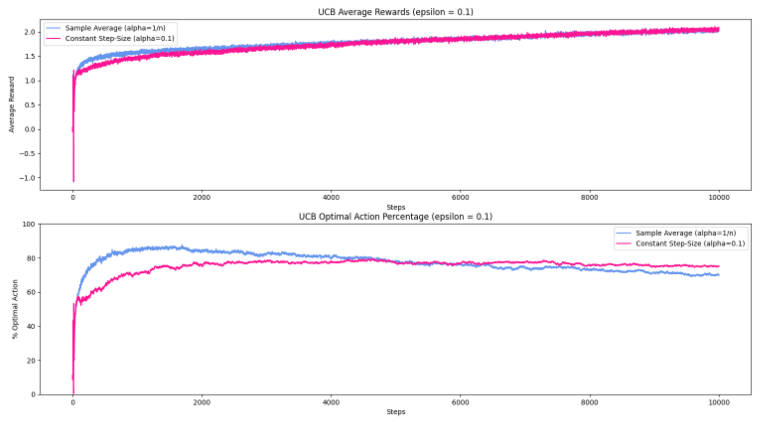

Figure 2: UCB results of average rewards for sample average and constant step size methods over 2,000 runs and 10,000 time steps. The above plot shows the average rewards, and the below plot shows the optimal action percentage of the two action-value methods.

We note that in both plots in Figure 2 that both methods start out at 0 at time step 0 and the sample average method is initially better than the constant step size. This follows the same reasoning as in Figure 1.

However, looking at the Average Reward plot, Figure 2 differs from Figure 1 in that the constant step size method did worse than the sample average method for longer. The two methods also basically have the same average reward values as each other, which was not expected. Both methods reach the same average reward of 2.0 at the end of the 10,000 time steps. The sample average was increasing at a higher rate than the constant step size, but then the constant step size caught up to it, most likely being due to how the sample average method is able to more accurately make decisions in earlier time steps when the highest rewards are in the beginning time steps.

Looking at the Optimal Action Percentage plot, the constant step size reaches and seems to maintain the same percentage as in Figure 1, which is 80 percent. The sample average method performed better than it did in Figure 1, seeming to plateau around 70 percent, so about 10 percent higher than what we observed in Figure 1. The sample average method was also much better for much longer than in Figure 1, which was unexpected.

This behavior could be due to how the sample average method leverages all past rewards when updating action-value estimates which would allow it to perform better in initial time steps, which is what we see. However, while these results do not support my assumption that epsilon-greedy would give better results than UCB, I do believe that Figure 2 supports the overall idea that over longer time stretches, that the constant step size method will ultimately perform better than sample average methods when the time steps become large enough.

5 Conclusion

The results from this experiment are useful in providing more information on the role that common action-value and action-selection methods can play when applied to non-stationary problems. From Figures 1 and 2, I would conclude that a constant step size method generally performs better than sample average methods for non-stationary problems. Based off of my results however, I cannot confidently conclude whether epsilon-greedy or UCB action selection is better for non-stationary tasks. The results from both action selection methods that I achieved are very similar and there is no definitive conclusion that I can draw about which one is better based off of Figures 1 and 2. I suspect that this is due to an error that I must have made in my implementation.

6 Extra Credit

For the extra credit portion of this assignment, I completed exercises 2.9 and 2.4 from Chapter 2 of Sutton and Barto. Both are non-programming tasks that were not covered in the live Zoom meetings nor the pre-recorded lectures.

6.1 Exercise 2.9

Exercises 2.9 states:

> Show that in the case of two actions, the soft-max distribution is the same as that given by the logistic, or sigmoid, function often used in statistics and artifical neural networks.

Solution:
The sigmoid function is given by:

$$sigmoid(x) = \delta(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

The softmax distribution is given by equation 2.11 in the book [1]:

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

We are considering the case of two actions. Let such two actions be represented as a_1 and a_2.

Plugging these two actions into the softmax distribution gives us:

$$P_r\{A_t = a_1\} = \frac{e^{H_t(a_1)}}{e^{H_t(a_1)} + e^{H_t(a_2)}}$$

we first multiply the numerator and denominator by $e^{H_t(-a_2)}$, so plugging in the negative of the second action, a_2, like so

$$= \left(\frac{e^{H_t(-a_2)}}{e^{H_t(-a_2)}}\right) \frac{e^{H_t(a_1)}}{e^{H_t(a_1)} + e^{H_t(a_2)}}$$

which simplifying by exponent rules results in

$$= \frac{e^{H_t(a_1)-H_t(a_2)}}{e^{H_t(a_1)-H_t(a_2)} + e^{H_t(a_2)-H_t(a_2)}}$$

$$= \frac{e^{H_t(a_1)-H_t(a_2)}}{e^{H_t(a_1)-H_t(a_2)} + e^{0}}$$

$$= \frac{e^{H_t(a_1)-H_t(a_2)}}{e^{H_t(a_1)-H_t(a_2)} + 1}$$

we note that if we let x = $H_t(a_1) - H_t(a_2)$ then this is equal to

$$= \frac{e^{x}}{e^{x} + 1} = \delta(x)$$

which is the sigmoid function.

Thus, we have shown that in the case of two actions, the soft-max distribution is the same as that given by the sigmoid function.

6.2 Exercise 2.4

Exercises 2.4 states:

> If the step-size parameters, n, are not constant, then the estimate
> Qn is a weighted average of previously received rewards with a
> weighting dierent from that given by (2.6). What is the weighting
> on each prior reward for the general case, analogous to (2.6), in terms
> of the sequence of step-size parameters?

We note that equation 2.6 [1] given in the book is:

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

$$... = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

Keeping this in mind, here is my solution:
The weights will be different with a non-constant α_n as now α_n will be varying at each time step.
This means that when we start out with the analogous 2.6 Q_{n+1} equation that we will be starting with α_n instead of just α since α_n is no longer constant:

$$Q_{n+1} = Q_n + \alpha_n \left[R_n - Q_n\right]$$

which we can rearrange to be

$$Q_{n+1} = \alpha_n R_n + (1 - \alpha_n) Q_n$$

we can then substitute Q_n with its update rule, which means that we are rewriting Q_n in terms of the prior time step $n - 1$. This is given by:

$$Q_n = Q_{n-1} + \alpha_{n-1}(R_{n-1} - Q_{n-1})$$

expanding we get

$$= Q_{n-1} + \alpha_{n-1}(R_{n-1}) - \alpha_{n-1}(Q_{n-1})$$

which can be rewritten as

$$= (1 - \alpha_{n-1})Q_{n-1} + \alpha_{n-1}R_{n-1}$$

substituting this back in for Q_n in the above Q_{n+1} equation gives us:

$$Q_{n+1} = \alpha_n R_n + (1 - \alpha_n)[(1 - \alpha_{n-1})Q_{n-1} + \alpha_{n-1}R_{n-1}]$$

expanding we get

$$= \alpha_n R_n + (1 - \alpha_n)(1 - \alpha_{n-1})Q_{n-1} + (1 - \alpha_n)\alpha_{n-1}R_{n-1}$$

we can continue to substitute using the same pattern that we have used so far. For example, Q_{n+1} can be expressed in terms of Q_n and Q_{n-1} can be expressed in terms of Q_{n-2}, etc.
If we follow this same recursive substitution pattern then we see that each R_n gets multiplied by α_n and each subsequent term is weighted by the previous updates made $(1 - \alpha_n)$. This results in:

$$Q_{n+1} = Q_1 \prod_{i=1}^{n}(1 - \alpha_i) + \sum_{i=1}^{n} \alpha_i R_i \prod_{j=i+1}^{n} (1 - \alpha_j)$$

From this analogous equation to equation 2.6 that we have derived, we see that the weight on each prior reward R_i is given by:

$$\alpha_i \prod_{j=i+1}^{n} (1 - \alpha_j)$$

References

[1] Richard S. Sutton and Barto Andrew G. *Reinforcement learning : an introduction.* The MIT Press, 2018.