A Comparative Analysis of Different Epsilon Values in a Multi-Arm Bandit Problem

Vincenzo Alessandro Macri[1]
Dept. of Electrical Engineering and Computer Science[1]
Florida Atlantic University
Boca Raton, Florida
vmacri2017@fau.edu

***Abstract*—Reinforcement learning is a form of unsupervised learning in the field of machine learning where an agent is trained to perform a specific task through trial and error. The agent is not given examples of how a problem should be solved, but instead is given a state and from that state must choose an action. Depending on how much the agent's action assists it on reaching the specified goal, the agent is given a reward and over time the agent will learn the correct actions to take, based on its current state, to maximise the reward it receives. The multi-arm Bandit problem is a classical benchmark problem in the field of reinforcement learning, where an agent is placed in an environment with multiple armed bandits. Each one of these arms, similar to slot machines in casinos, when pulled has a certain probability of returning a specified reward. Through methods of calculating expected rewards over time, and trading off between exploratory and exploitative actions, the agent is tasked with correctly identifying the most bountiful arm to pull with the goal of maximising its total collected reward. This paper will discuss the algorithms and methods used to govern such an agent and delve into the exploration–exploitation trade-off dilemma. The paper will conclude with demonstrations of real-world applications of this problem and visualize the training process of such agents through plots and figures.**

***Index Terms*—reinforcement learning, multi-arm bandit, exploratory actions, exploitative actions, python**

I. Introduction

The multi-arm Bandit problem takes place in an environment with multiple armed bandits that each, similar to slot machines in casinos, when pulled have certain probabilities of returning specified rewards. In this paper we conducted two experiments using this environment framework where a reinforcement learning agent, governed by a *Q*-value based update function is tasked with learning the best arm to pull to maximize its average reward over time, and in doing so, maximize its total reward at the end of the simulation. When making a decision on which arm to pull, the agent will either choose a random arm, known as an exploratory action, with a probability denoted by a constant value ε, or choose the arm it expects the highest reward from based on previously learned knowledge with a probability of $1-\varepsilon$. In both experiments, we analyzed and compared the performance of the reinforcement learning agent when using the epsilon values 0, 0.01, and 0.1 to govern the ratio of exploratory and exploitative actions to see which value for ε produces the best model.

II. Algorithm

A. Involved Formulas

In both experiments the *Q*-value estimation formula (1) was used to update the agent's *Q*-values after each iteration step. Each arm has a corresponding *Q*-value which holds an estimated reward value. Each time the agent chooses an arm, that arms corresponding *Q*-value is updated following the formula (1) where Q_{n+1} is the updated *Q*-value for the next iteration, Q_n is the *Q*-value as of the current iteration, n is the current iteration index and R_n is the actual reward received by the agent based on its action in the current iteration.

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n] \tag{1}$$

B. Pseudo code

The following psuedocode describes the main algorithm used each time a simulation is run during our experiments with iteration and epsilon values entered as inputs. This algorithm is then run multiple times in an outer loop with varying input values to average out any extreme outlying results. The written psuedocode is based off of Sutton and Barto's 2018 textbook *Reinforcement Learning: An Introduction* for reference [1].

Algorithm 1 Simple Multi-armed Bandit Algorithm

Initialize, for a = 1 to k:
$Q_a \leftarrow 0$
$N_a \leftarrow 0$
loop for each iteration:
$A = \begin{cases} argmax_a Q(a) & \text{with probability } \varepsilon - 1 \\ \text{random selection} & \text{with probability } \varepsilon \end{cases}$
$R \leftarrow$ reward received from A
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$
end loop

C. Parameters and Setup

For both experiments, we used python 3 in the Google Colab environment to implement our setup and algorithms. We used the libraries numpy, matplotlib and pandas to assist with array manipulation, graphing and file manipulaiton respectivly in our program.

1) Experiment 1: In experiment 1, we begin by using ten random values, selected using a normal (Gaussian) distribution with mean 0 and variance 1, as our true reward values for the ten arms, each denoted by $q_*(a)$. The probability of collecting a reward when a given arm is pulled by the agent is simulated in an analog fashion by using a normal distribution with mean at $q_*(a)$ for each arm and variance 1. A random sample of these reward distributions can be visualized by the 10 grey shaded regions illustrated in [1, Fig. 1]

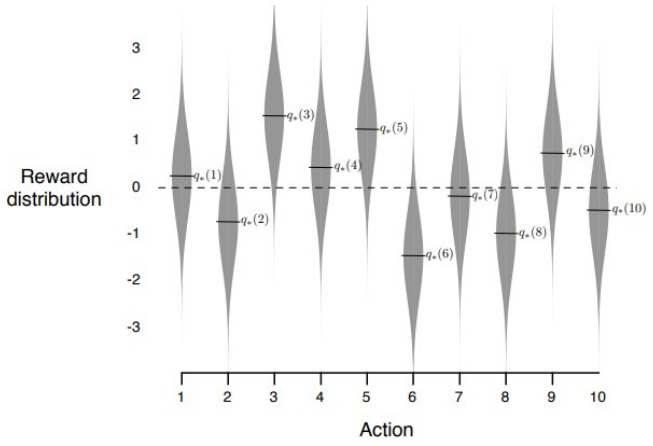

Fig. 1. Sample rewards distribution used in Experiment 1

We ran the model using three different values for ε; 0.0, 0.01 and 0.1. For each of these trials, there is an outer loop and an inner loop. Inside of the inner loop, 10 arms are selected using the procedure described above, and the loop runs for 10,000 iteration steps with the agent updating its expected *Q*-values after each iteration. In the outer loop, this inner loop process is conducted 1,000 times, each with different arm values. Using an outer loop to run the inner simulation 1,000 times, eliminates the possibility of skewed results that could be strictly dependent on specific arm values for a single simulation. After the outer loop was run for all three ε values, we plotted the average reward collected by the agent at each time step and the percent of optimal actions taken by the agent at each time step of the simulation.

2) Experiment 2: In experiment 2, rather than generating our own values for each arm's return reward, we are given a CSV file with 10,000 rows of 10 columns. Experiment 2 is meant to reflect a real world scenario and use case for the multi-armed bandit problem where a hypothetical advertising company is running 10 different ads targeted towards a similar population set on a web page. Each column index of the CSV file represents a different ad and each row index of the CSV file represents a user. For each entry in the file, there is a 1 if the ad was clicked by a user, and 0 if it was not. Using this data, during each iteration we can set the reward collected by the agent for selecting a specific ad, equal to the value stored in the element at that ad's index and current user row index. Similar to experiment 1, we ran the model using three different values for ε; 0.0, 0.01 and 0.1. For each of these trials we had an inner loop which ran for 10,000 iterations, enabling the agent to traverse the entire data set, and an outer loop which ran 1,000 times. Before each of the 1,000 runs of the inner loop, we used a function from the pandas python library to shuffle the rows of the CSV file around to eliminate any biases presented in the order of the data. After the outer loop is run for all three ε values, we plotted the average reward collected by the agent at each time step, the total running reward collected by the agent at each time step and the percent of optimal actions taken by the agent at each time step of the simulation.

III. Results and Discussion

In figures (2), (3), (4), (5) and (6) which illustrate results from both experiments 1 and 2, we took average results over time, both in the inner and outer loops of our algorithm in order to smooth the noise that comes with performing 1,000 outer loop steps, and to better visualize the trajectory of each curve.

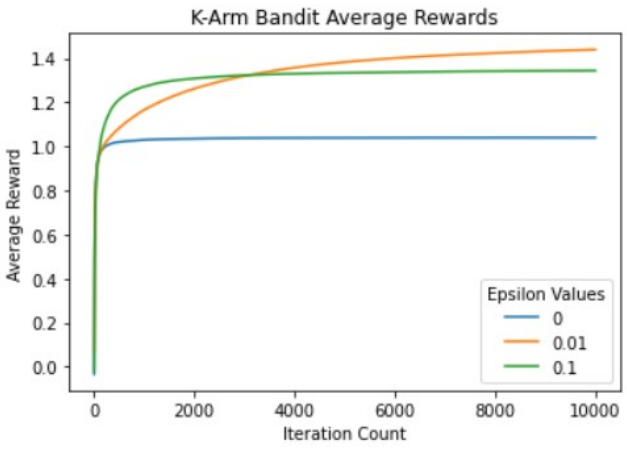

Fig. 2. Average reward received by agent over time in Experiment 1

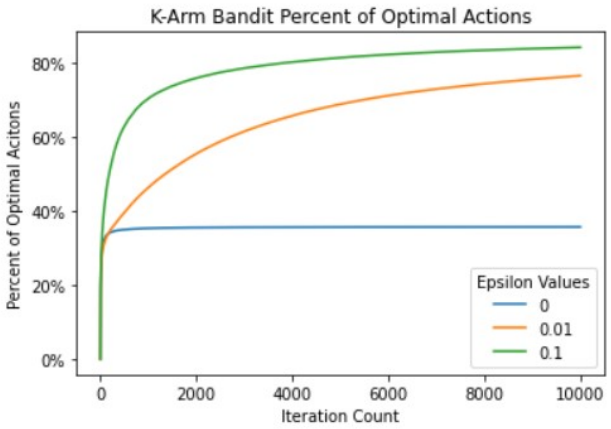

Fig. 3. Percent of optimal actions taken over time in Experiment 1

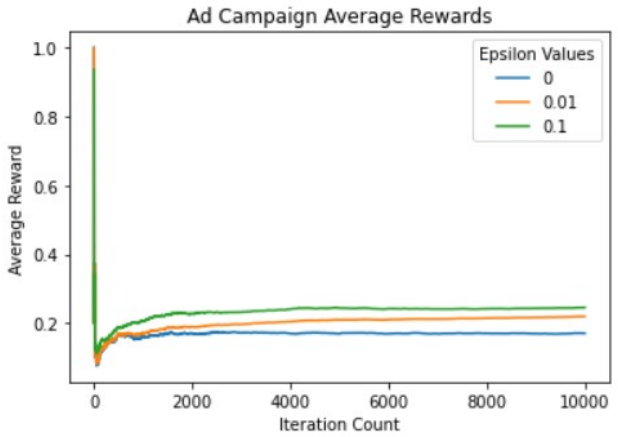

Fig. 4. Average reward received by agent over time in Experiment 2

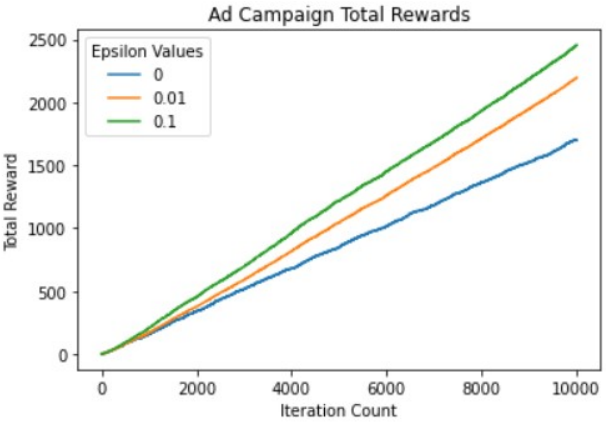

Fig. 5. Total running reward received by agent over time in Experiment 2

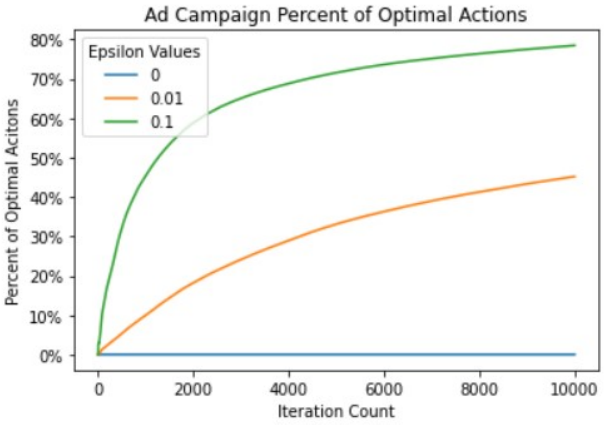

Fig. 6. Percent of optimal actions taken over time in Experiment 2

A. Experiment 1

1) Average Reward over Time: After conducting experiment 1, we could see from figure (2) that by the end of the 10,000 iterations, the model using $\varepsilon = 0.01$ out performed the other two models with an ending average reward of 1.45. The model using $\varepsilon = 0.1$ followed short behind with an ending average reward just under 1.3 and the model using $\varepsilon = 0$ ended the simulation with an average reward value of 1.0. The model using $\varepsilon = 0$ finishing the simulation with an average reward value well below the two other models seems logical since the model will never select arms at random and explore new opportunities. This in turn will cause the agent to get stuck at the arm it initially chooses at random. It is interesting, however, to notice the differences in slope of the trajectories of the models using $\varepsilon = 0.01$ and $\varepsilon = 0.1$. The model governed by $\varepsilon = 0.1$ initially takes the lead in terms of average reward over time with a higher slope that tapers out relatively quickly after around 2,000 iterations. This behavior can be attributed to the relatively high rate of exploratory actions at 10 percent of all actions taken by the agent. At first, by having such a high rate of exploration, the agent is able to discover the optimal arm relatively quickly since it can cover more ground in a swift time frame, explaining the swift jump in average reward over time in the earlier iteration steps. However, this model will never be able to pick the most optimal arm more than 90 percent of the time, since even after the model converges on the correct value for the most optimal arm, it will still pick an arm at random 10 percent of the time. On the other hand, the model governed by $\varepsilon = 0.01$ has a relatively small slope throughout all of the iteration steps, however it is more stable and does not taper out quickly like the model governed by $\varepsilon = 0.1$ does. This behavior can be attributed to the lower rate of exploratory actions at only 1 percent of all actions taken by the agent. Since the exploratory rate of the agent in this model is so low, it may take the model longer to discover the most optimal arm to pull as it is given less opportunities to try out different options. However, once the model converges on the correct arm, this disadvantage becomes an advantageous trait for the model as the agent will pick the optimal arm 99 percent of the time and only waste time picking a random action one percent of the time. This in turn leads to the model governed by $\varepsilon = 0.01$ surpassing the model governed by $\varepsilon = 0.1$ in terms of average rewards collected overtime, with higher iteration index values.

2) Percent of Optimal Actions: Figure (3) shows the effects of different ε values when used to govern models in experiment 1 in terms of the percent of optimal actions achieved by each of the models over time. To calculate the optimal action for each inner loop simulation, we took the arm with maximum randomly generated mean value. As expected, the model using $\varepsilon = 0$, returned a low and constant rate of optimal action selection, hovering around 35 percent. Interestingly however, the model governed by $\varepsilon = 0.01$ did not surpass the model governed by $\varepsilon = 0.1$ as it did in figure (2) but instead only reached the optimal outcome 70 percent of the time on average by iteration 10,000 where as the model governed by $\varepsilon = 0.1$ excelled to reaching the optimal outcome on average 80 percent of the time by iteration 10,000. While at first, this may seem unreasonable, it logically makes sense that the model governed by $\varepsilon = 0.1$ excels in this metric when we

take into account how the speed of growth effects the total average percent of optimal actions chosen over time. Even though the model governed by ε = 0.01 excels at reaching the highest average reward by the end of the simulation, its slow exploration rate results in the model getting stuck on non-optimal arm values by random chance for many iterations at the beginning of the simulation where as the model governed by ε = 0.1 can begin choosing optimal actions more frequently at a quicker rate. Even though, eventually, the model governed by ε = 0.01 will pick the optimal action more frequently by the latter half of the simulation, its initial blunders make it more difficult for new data to adjust the average value for its percent of optimal actions chosen. Eventually however, given enough iterations, we believe the model governed by ε = 0.01 will eventually surpass the model governed by ε = 0.1 in the metric of its percent of optimal arms chosen based on its trajectory.

B. Experiment 2

1) Average Reward over Time: After conducting experiment 2, we can see from figure (4) that the model governed by ε = 0.1 achieved the highest average reward by the end of the simulation at 0.24, followed by the model governed by ε = 0.01 at 0.21 and finally the model governed by ε = 0 at 0.17. When looking at the graph of these trajectories it is very obvious that there is a quick jump to 1.0 as the average reward at the very beginning of the simulation for all three models that is then followed by a rapid decrease to more normalized values. The reason this occurs is because the agent in these models either receives a 1 or 0 as its reward when choosing an ad, and at the beginning of the simulation, just by random chance, the agent may receive a reward of 1. If, for example, this occurs at the first time step, the average reward would be calculated to be $\frac{1}{1} = 1$ or if it occurs instead at the second time step, the average reward would be calculated to be $\frac{1}{2} = 0.5$ in consequence giving false impressions of the true nature of the model. With this fact aside, looking at the rest of the models trajectories, it looks as though the model governed by ε = 0.01 is slowly climbing and on track to eventually surpass the model governed by ε = 0.1, however for the scope of this simulation it was not given enough iterations. We believe this behaviour is similar to what is observed in the first 2,000 iterations of figure (2) in experiment 1. Additionally, we hypothesize that one of the reasons the model governed by ε = 0.1 is able to outperform model governed by ε = 0.01 is due to the nature of the data set. Since there are certain rows of users that return a reward of 1 for multiple ads at the same time, it is not as much of an issue if the agent picks at random 10 percent of the time after convergence since there is still a likely chance to get a reward of 1 from a random action.

2) Total Reward over Time: Figures (4) and (5) are closely related since figure (5) visualizes the total running reward collected by each models agent overtime where as figure (4) takes the total reward values at each time step and averages them out over the total iterations done at that time step. Similar to the ranking of models in the analysis of figure (4), the model governed by ε = 0.1 achieved the highest total reward by the end of the simulation with a total reward of 2454.05, followed by the model governed by ε = 0.01 with a total reward of 2195.23 and finally the model governed by ε = 0 at 1703.0. These values are directly proportional to the values described by figure (4), therefore the reasoning for these results follows the same analysis described in the previous subsection. Based off of the agents selections in the best performing model, the most successful ad in the campaign was chosen as Ad 5.

3) Percent of Optimal Actions: To determine the true optimal ad for experiment 2, for each ad column, we took a summation of all 10,000 rows in the ad data set and found that Ad 5 was the most successful ad in the campaign with a total of 2695 user clicks. Using this measure, we were able to compute the percent of optimal actions taken by each models agent over time. Figure (6) shows the effects of different ε values when used to govern models in experiment 2 in terms of the percent of optimal actions achieved by each of the models over time. The model governed by ε = 0.1 performs the best in this metric, choosing the optimal action an average of 80 percent of the time by iteration 10,000. The model governed by ε = 0.01 only reached choosing the optimal action an average of 40 percent of the time by iteration 10,000 where as the model governed by ε = 0 remained near 0 percent in this metric the entire simulation. The reason for this behavior can be understood when we take into account how the speed of growth effects the total average percent of optimal actions chosen over time as we discussed in subsection 2 for experiment 1.

IV. Conclusion

In short we can conclude from both experiments that there is a clear trend in the behavior of the slopes of multi-armed bandit models governed by ε = 0.1 and ε = 0.01. In models governed by ε = 0.1, the average reward collected by an agent increases more rapidly due to the higher rate of exploration towards the beginning of a simulation, and tappers off when it converges to the most optimal outcome. In contrast, for models governed by ε = 0.01, the average reward collected by an agent increases at a slower, more continuous rate as the agent's rate of exploration is lower. However, we have seen that this allows models governed by ε = 0.01 to out perform models governed by ε = 0.1 in the long-run since models governed by ε = 0.01 can continue climbing in the metric of average rewards past where a model governed by ε = 0.1 can once the agent converges on the optimal arm in its environment. Finally, we found in experiment 2 that the model governed by ε = 0.1 was able to achieve a maximum total reward of 2454.05 after it converged to the correct, most successful ad, Ad 5. Both of these experiments showed the power of reinforcement learning and the effects the value of ε has on a model.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction 2nd ed*, 2018.