Reinforcement Learning Assignment 1

Levente Foldesi (s3980456), Samuele Milanese (s3725294)

December 9, 2021

1 Introduction

For this assignment we experimented and compared the efficacy of different Action-Value methods in solving the classic multi-armed bandit problem. The code base was developed in Python 3 with the help of basic libraries such as numpy, random and matplotlibpy.

2 Algorithms

2.1 Greedy & Epsilon-Greedy

The greedy action selection was based on the following equation: $A_t = argmax(Q_t(a))$. Where $Q_t(a)$ is the action value which is the reward we expect to receive by taking that action in time step t. The greedy algorithm looking for the maximum of this action value (maximum reward).
Epsilon-Greedy works similarly, the only difference is that it adds a little twist to the greediness. There is a chance (ϵ) that the algorithm will explore an other action randomly instead of exploiting the already found actions in order to "get out" of the potential local maximum.

$$\pi_t(a) = \begin{cases} (1-\epsilon) + \epsilon/|A| & a = argmax(Q_t(a)) \\ \epsilon/|A| & \text{otherwise} \end{cases}$$

Where A is the set of possible actions and ϵ is between 0 and 1.

2.2 Optimistic initial values

In our implementation we used the greedy algorithm as a base to test the efficacy of optimistic initialization. Hence, the only difference with the normal greedy algorithm is that the values are initially set to a fixed value other than 0 so to encourage exploration. We set $Q(a) = 50.0$ for all actions in our algorithm to achieve the aforementioned goal.

2.3 Upper confidence bound

For the UCB algorithm we used the following equation: $a_t = argmax[Q_t(a) + c\sqrt{\frac{ln(t)}{N(t)}}]$ where c is a constant called confidence value which controls the level of exploration, t is the current time step and $N(t)$ is the number of time an action has been selected prior to t. The way this equation is represented in the algorithm is that $Q_t(a)$ is the part for exploitation and $c\sqrt{\frac{ln(t)}{N(t)}}$ is the part for exploration. In the beginning, the algorithm will chose exploration more often but later on, as the highest estimated reward ($Q(a)$) is getting higher, it will rather exploit.

2.4 Softmax

Softmax algorithm is based on a probability which indicates the chances of an arm being pulled. This probability is calculated as the following: $P(\text{Explore arm j}) = \frac{e^{r_i/\tau}}{\sum e^{r_k/\tau}}$ Here r_i represent the average reward for the current arm i, τ is a hyperparameter which decides the level of exploration. If τ is close to one that would give more chance for exploration so conversely, if it is closer to zero then the chance for exploitation is higher.

2.5 Action preferences

This method relies on learning a certain preference for an action $H(a)$ (where the higher the preference is for one arm the higher the chance that the arm is going to get chosen). Then for calculating the policy we used the Softmax distribution (as used in the previous algorithm): $\pi(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{n} e^{H_t(b)}}$ This will make a list of probabilities for each action in time t (this is notated by $\pi(a)$). Then for keep updating these probabilities, the following equation was used: $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \overline{R}_t)(1 - \pi_t(A_t))$. Where A_t is the action in time t, α is a step function and $\pi_t(A_t)$ is probability of choosing action in time t.

3 Experiment design

Our code structure is formed by three elements: The bandit class which has a random generator that will hand out the reward to the agents; the agents which are all designed in terms of the same blueprint but implement the different algorithms involved in the experiments; finally, the main loop which practically executes the experiment.
Each run goes as follows: all the algorithms run for N times and complete a full round of experiment where they learn the policy π, each following its strategy. At the end of this round, the statistics about reward and optimal solution per each time-step are collected. Then the agents are reset and the experiment is repeated N more times.
In each run each model selects one of the k bandits available following its strategy and receive their reward. The reward can differ based on which scenario of the problem we are running. We could either have a reward which is sampled from a random and unique Gaussian distribution which mean and standard deviation are randomly generated by a small interval governed by some hyper-parameter or a binomial reward which returns 1 or 0 with probability p.
Although the setup is quite simple it still comprises several hyper-parameters which can change drastically the results of the experiments.
Some hyper-parameters are concerned with the setup itself such as N and K. N controls the number of steps in one iteration and the number of iterations of the experiment as well. On the other hand, K controls how many bandits are in the problem, so how many arms the agent must decide among to get the reward. We chose a value of 1000 for N so that the experiment could repeated enough times to give good averages, while we chose 4 arms bandits as these gave fairly good results since, a high k often meant the problem was 'too difficult' for the agents.
Other hyper-parameters govern the distributions behind the rewards. These are the MU and SIGMA intervals. When a bandit is initialized it gets a random distribution from which to sample the rewards with mu and sigma sampled from these intervals. We chose relatively small intervals for both parameter, the mean parameter was set to (4,8) (since a higher mean tend to yield better results but for greedy a too high mean was disadvantages so we found this optimal solution). The sigma parameter, however was set to (0,1). The reason for this was to create a low margin between the optimal solution and the least optimal solution, this way even if the algorithm does not find the optimal solution, it can still have a high reward. It also turned out that with a high mean and a low standard deviation, the precision of the algorithm was increase significantly.
Finally we have parameters involved with the equations of the algorithms themselves. These were tuned as a result of both experimenting and sensible value suggested by theory. the Epsilon parameter determined the inclination of the epsilon greedy algorithm to explore and we set it to 0.1. This value is very common in textbooks, and we have proved ourselves to be the best performing value. The tau parameter also determines the eagerness of the Softmax algorithm to explore and was set to 0.9 which gives plenty of room for exploration. The Qa parameter corresponds to the initial value to which all values are set to for the optimistic initialization algorithm. This parameter does not really need testing as it is sufficient to set it to a very high number (at least just more than the biggest possible reward) for it to achieve its purpose. The c parameter concerns the UCB algorithm and represents the influence of uncertainty on the decision of the algorithm. After testing we set it to 3.0. Finally we have the alpha value for the Action Preference algorithm, this mitigates the influence of the rewards received up until that time-step. We set it to 0.1.

4 Plots

We included four plots to support our findings in our discussion. The first two plots is the result with the Gaussian reward function, whereas the second two is with the Bernoulli reward function.

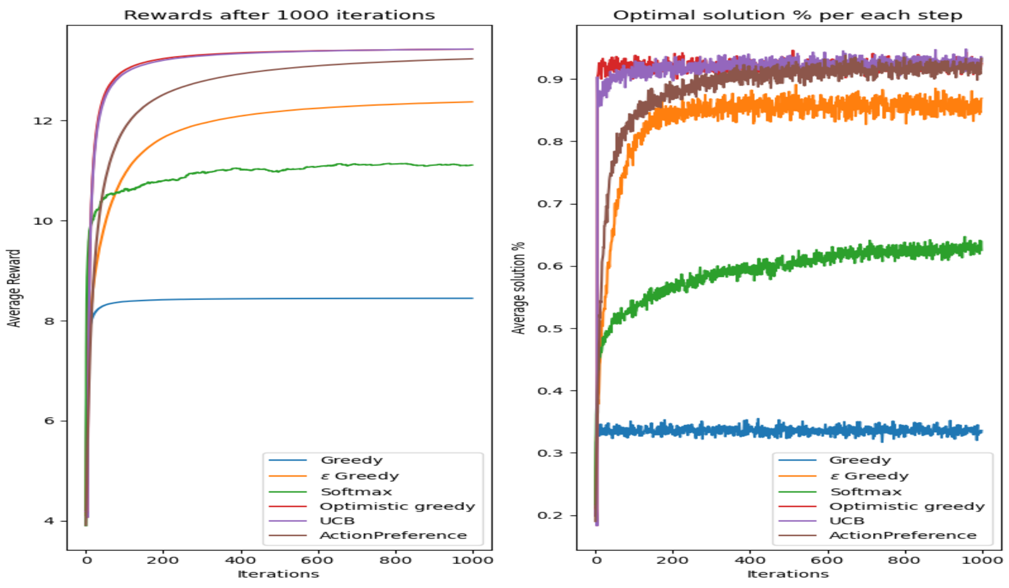

Figure 1: Results of running the experiment with a Gaussian distribution and the following parameters: N = 1000, K =5, epsilon = 0.1, tau = 0.9, Qa = 50.0, c = 3.0, alpha = 0.1

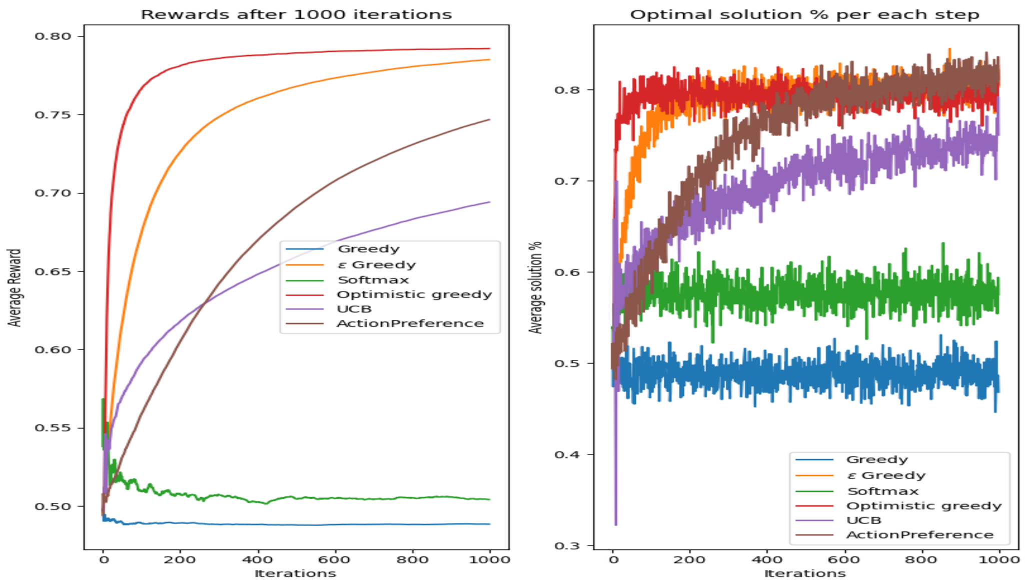

Figure 2: Results of running the experiment with a Bernoulli distribution and the following parameters: N = 1000, K =5, epsilon = 0.1, tau = 0.9, Qa = 50.0, c = 3.0, alpha = 0.1

5 Discussion

Overall, it can be said that the different reward function yield different results regarding which algorithm performs best. It is also seems that most of the algorithms performed better than 50% regardless of the reward distribution (the only exception was greedy with the Gaussian reward function). In the following two paragraphs we will talk about each algorithms with respect to performance in both reward functions.
Regarding the performance of different algorithms in the Gaussian version. There are two outliers which seems to be significant, one of them is Greedy which is quite straight forward since it is the easiest and simplest version. Since it has the same "tactics" throughout all the steps it make sense that it has the worst performance. The other one is Softmax which still perform better than 60% but way worse than the others. The reason for that can be that τ which indicates whether to explore or exploit more remains constant (at least in our implementation) which can lead to worse performance compared to a version when τ is declined over time. The other four algorithms were almost similarly successful however, ϵ-Greedy was a bit under the other three. As it is still mainly based on greediness and we did not use an optimistic initial value (which could have made most of the algorithms better but we decided to just compare the algorithms without it as this shows how can each algorithm perform without "fusion"), this also makes sense that it performs better than greedy because of the chance of exploring but worse than others.
The Bernoulli version yield a bit different results. The two outliers is the same here as well for the same reasons as in the aforementioned paragraph. The main contrast is that now the order of the best four algorithm is changed in a sense that now UCB is a bit worse than the others. This could be a consequence of the absolute value of the reward being way smaller in the Bernoulli version (at most 1). This makes it so that in the computation of the UCB the exploration part weighs more and thus the algorithm tends to explore rather than exploit.
In summary, it can be said that Initial optimistic values and Action preferences algorithms were the most flexible ones since both performed extraordinarily, regardless the reward function, while greedy and Softmax seemed to be the least successful ones.

6 References

https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f

https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-softmax-algorithm-e1fa4cb0c422

https://towardsdatascience.com/multi-armed-bandits-and-reinforcement-learning-dc9001dcb8da