ASSIGNMENT 1: EXPLORATION/EXPLOITATION ON BANDITS

Student names **Nataliia Bagan, Maksym Lytovka**
Student numbers s3648885, s3705609

1 INTRODUCTION

The problem considered in the present assignment is k-armed bandits, with $k = 10$. It can be described with an example of slot machines. An agent is faced with a 10-armed machine and has a certain number of times to play. For each arm, the reward follows the Bernoulli distribution with a random mean, and the mean itself is sampled from the uniform distribution on $[0, 1]$. To put it formally: $r_a \in \{0, 1\}$, $r_a \sim \text{Bernoulli}(\mu_a)$, $\mu_a \sim \text{Uniform}(0, 1)$, where r_a is a reward for a chosen action (arm) a and μ_a is a probability of getting the reward spinning the arm a. Agent's objective is to choose the arms in such a way as to achieve the highest average (over all spins) or cumulative reward - depending on the formulation.
In the present report, we formulate it in terms of a simplified non-sequential version of MDP, as described in Chapter 2 of (Sutton & Barto, 2018). The paper will follow the following notation: A - set of actions ($|A| = 10$), a and b - certain actions ($a, b \in A$, in out implementation: $a, b \in \mathbb{N}$), $Q(a)$ - the current estimated value of action a ($Q : A \to \mathbb{R}$). Other appropriate notations will be introduced in the corresponding sections. We focus on the following 3 policies and look at their performance in the problem formulated above: 1) ϵ-greedy, 2) Optimal initialization (OI), 3) Upper confidence bound (UCB).
In sections 2-4 we examine the policies by themselves, looking for the best hyperparameters. 3 features characterize each policy: 1) initial values (optimistic/realistic), 2) policy (the 3 named above), 3) update rule for the means (incremental / learning-based). Policy parameters are compared in 2 ways, which adjust to the needed results: whether we are interested in high immediate rewards, or high average reward of the run. Therefore, firstly, we compare algorithms by the reward achieved on each of 1000 steps, averaged over 500 repetitions, and plot the results. And secondly, we observe average reward achieved before and up to each of 1000 time steps (current cumulative reward divided by current step), averaged over 500 repetitions. We will call the former "learning" curves, and the latter "performance" curves. After separate comparisons, we compare the optimized policies with each other in section 5. It is done firstly, by considering average reward over 1000 steps of each algorithm and hyperparameter; and secondly, based on the best found values for parameters in the previous sections.

2 ϵ-GREEDY

2.1 METHODOLOGY

The first policy considered is an adjusted version of a greedy policy, which accounts for exploration compared to the original one (which purely picks action with highest expected return every time). Epsilon ($\epsilon \in [0, 1]$, $\in \mathbb{R}$)) is the parameter of exploration: with a chance of $1 - \epsilon$ the policy exploits the best action, and otherwise explores (with the same probability of being chosen assigned to each non-greedy action). Taken initial values in our implementation are realistic, to later have a more clear comparison with algorithm of optimistic initialization. Among the 2 update rules, proposed in the introduction, incremental is chosen. Therefore, implemented algorithm is the following:

1. Initialize the estimates for the mean $Q(a)$ of each action to 0. Initialize counts $n(a)$ of each action to 0 too, for the incremental update (each $n(a)$ stores how many times the action in the argument has been chosen already).
2. On every step select the action based on ϵ and following formula of the policy (although, another version of the formula is available, which distributes exporation among all actions. See why it is

more mathematically appropriate in Appendix7.1):

$$\pi_{\epsilon-greedy}(a) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_{b \in A} Q(b) \\ \frac{\epsilon}{|A|-1}, & \text{otherwise} \end{cases} \tag{1}$$

3. Update the mean with an incremental update rule:

$$n(a) \leftarrow n(a) + 1, \qquad Q(a) \leftarrow Q(a) + \frac{1}{n(a)}[r(a) - Q(a)] \tag{2}$$

, where $n(a) \in \mathbb{N}$, as mentioned, is equal to number of times arm a has been played. $r(a) \in \mathbb{R}$ is equal to observed reward for the taken action a.
We then compare reward (or average reward) curves averaged over 500 repetitions of the agent learning, each curve corresponding to one of parameter variants: $\{0.01, 0.05, 0.1, 0.25\}$. By including both learning and performance curves we can see how each version is performing both on each time step separately, and overall (the average reward recieved by the agent so far). Such choice of epsilon values provides sufficiently various settings, yet we ignore 1) $\epsilon = 0$ (greedy policy) to always have exploration to some extent; and 2) any options where exploitation is too little (in our case < 0.75). This way, we do not consider redundant curves, as these values do not bring any more improvement on the algorithm (see Appendix 7.5.1 for more detailed explanation).

2.2 Results

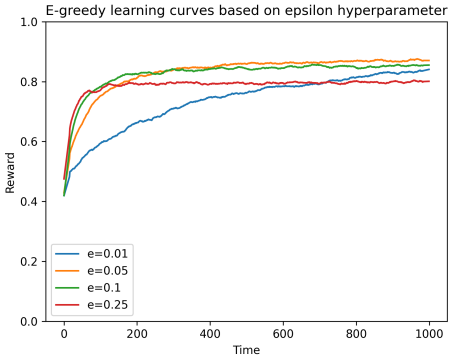

Figure 1.1: Corresponding learning curves (each point is immediate reward averaged over repetitions).

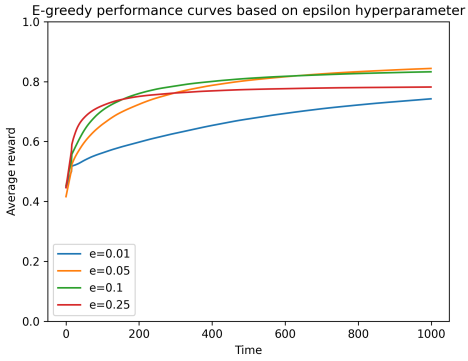

Figure 1.2: Corresponding performance curves (each point is average reward of the run up to curent time step, averaged over repetitions).

Figure 1: Effect of exploration parameter epsilon on performance of ϵ-greedy policy over 1000 time steps. Reward (left) and average reward (right) at each time step is averaged over 500 repetitions, curves are smoothed with smoothing window equal to 31. The number of possible actions is 10. Among the 4 considered values of ϵ, the best curves are at $\epsilon = 0.1$ and 0.05. 0.01 learns too slow at the beginning, and 0.25 reaches lower values in the end.

The figure above shows the comparison of the chosen ϵ values. We can see from the Figure 1.2 that even though 0.01 learns slower than 0.25 and has lower average reward at all time steps (Figure 1.2), it reaches higher value toward the end (Figure 1.1), making it better for longer runs. Although, the best parameter for our problem is either 0.05 or 0.1 and it depends on the number of time steps and measurement of performance (reward on Figure 1.1, or average reward on Figure 1.2). As can be seen, the curve of $\epsilon = 0.1$ leaps faster at the first time steps (similar to 0.25, because it explores best action earlier in the run), but later exploits the best action less, resulting in a bit lower final immediate rewards, comparing to 0.05. Therefore, it would the preference if we have small number of time steps, or our measurement is average reward. Otherwise, 0.05 achieves higher immediate rewards at bigger time steps, and it would be appropriate choice. Overall, both parameters perform similarly well, and the main point we can see form the plot is that more extreme values either explore too slow at the beginning, or exploit too little at the end: intermediate settings result in better learning and performance curves overall.

3 Optimistic Initialization

3.1 Methodology

The second policy we discuss is the Optimistic Initialization with greedy action selection (further OI policy). This policy is very similar to a simple greedy policy. The main difference is that we initialize the means to be some non-zero value (usually positive and relatively high), in contrast as we assign 0 to every mean at the beginning of the plain greedy policy. It allows an agent to explore the options before sticking to some particular one. By setting the high opening value, we decrease the mean at every step. Therefore, as the policy is still greedy, we choose the action, which mean is still the highest. The higher the initial value, the more time we give for the agent to explore. That is why we need to tune this value to our specific problem and its ranges. Hereby, the learning algorithm is the following:

1. Initialize the estimates for the mean $Q(a)$ of each arm to some initial value k.
2. On every step select the action with the highest mean (policy is greedy):

$$\pi_{greedy}(a) = \begin{cases} 1, & \text{if } a = \arg\max_{b \in A} Q(b) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

3. Update the mean with a learning-based update rule for the mean:

$$Q(a) \leftarrow Q(a) + \alpha[r(a) - Q(a)] \tag{4}$$

, where α is a learning rate. $\alpha \in (0, 1], \in \mathbb{R}$. In the present experiment we fix it to 0.1. However, it is important to mention, that we use that learning rate only in this policy (both ϵ-greedy and UCB use multiplier of $1/n(a)$ for the update). It is done with the purpose, as with the constant multiplier the previous numbers will gradually "fade away" and we only "remember" the last numbers. Meanwhile, the $1/n(a)$ multiplier will gradually decline, making us treat all the previous rewards equally. As in the analysis of previous algorithm, we make 2 plots, learning and performance, averaged over 500 repetitions. We also smooth them for more clear visual results. Considered init_val parameters are: $\{0.1, 0.5, 1.0, 2.0\}$. Such minimal and maximal values are enough to cover algorithm boundaries in the current setup of the problem (explained in detail in Appendix 7.5.2).

3.2 Results

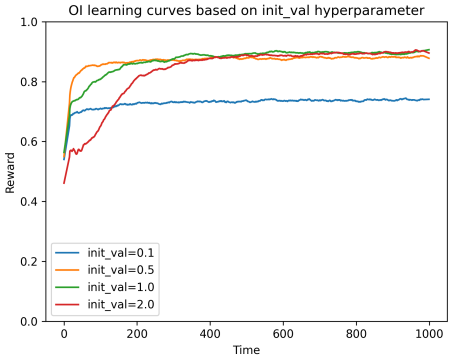

Figure 2.1: Corresponding learning curves (each point is immediate reward averaged over repetitions).

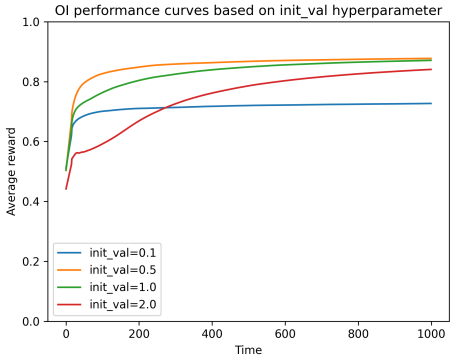

Figure 2.2: Corresponding performance curves (each point is average reward of the run before time step, averaged over repetitions).

Figure 2: Effect of different initial values of $Q(a)$ on performance of OI (+greedy) policy over 1000 time steps.Reward (left) and average reward (right) at each time step is averaged over 500 repetitions, curves are smoothed with smoothing window equal to 31. The number of possible actions is 10. Among the 4 considered init_val settings, 0.1, 0.5, 1.0 reach similar final rewards, with $= 0.5$ having the fastest leap at the beginning (but a bit lower final reward) and 2.0 - the slowest, which also impacts average reward (right). 2.0 also has lowest first point of the curve. 0.1 is a clear underestimate as in 1000 steps it still does not approach other curves.

With the described methodology in mind, we conduct the experiment, looking for the best initial value. Figure 2 shows the results of the experiment. From the figure we clearly see that extreme values are not preferable. To make OI policy work best, we have to set the initial value neither too high, nor too low, so that we catch the balance between exploration (a jump at the first time steps, Figure 2.1) and exploitation (values at final steps, Figure 2.1). Moreover, we see, that the curves representing the high initial values still converge to the best line (which is init_val $= 0.5$), however, running behind it (especially considering average reward on Figure 2.2). That may be explained by the idea, that these agents were exploring for a longer, because they gradually decreased from a larger init_val, that is why the gap was formed. In theory, if more time steps are given, all three agents with init_val ≥ 0.5 will perform almost equally well at the end. Hereby, when tuning the initial value hyperparamenter, the available number of time steps should be taken into the consideration.

4 Upper Confidence Bounds

4.1 Methodology

The last policy considered is the Upper Confidence Bounds policy (further UCB policy). The main idea of this approach is to consider the confidence interval instead of the mean values. It goes simple: for every action we calculate the mean μ and the value δ - how much the mean may vary (in two sides). Then, we take the upper bound, which equals $\mu + \delta$. Comparing the upper bounds, we select the action with the highest one. The value of δ depends on the number of times we have selected the present action already, and the time step. It is quite intuitive that if you have select some action a 10 times and action b 2 times, and both of them give you the reward every second time, you will be more sure, that the probability of a is 0.5, as you have tried it more. That is why δ depends on number of times you performed the action. The initial values of the means for every action are 0, just like in the ϵ-greedy policy. Thus, we have the algorithm:

1. Initialize the estimates for the mean $Q(a)$ of each arm and counts $n(a)$ to the initial value of 0.
2. On every step select the action with the highest upper bound:

$$\pi_{UCB}(a) = \begin{cases} 1, & \text{if } a = \arg\max_{b \in A} \left[Q(b) + c \cdot \sqrt{\frac{\ln t}{n(b)}} \right] \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

, where t is the time step ($t \in \mathbb{N}$), $n(b)$ is the number of times we have performed action b (same $n(a)$), and $c \in \mathbb{R}^+$ is the exploration constant (our parameter of interest taht will be tuned). The term $c \cdot \sqrt{\frac{\ln t}{n(b)}}$ in the equation implies the δ we have discussed above.

3. For updating, use the incremental update rule for the mean, same as in section 2.2, formula (2).

Considered hyperparameter values are the following: $\{0.01, 0.05, 0.1, 0.25, 0.5, 1.0\}$. Appendix 7.5.3 shows that these values are representative. Setup of the experiment stays the same, we plot immediate and average rewards, (additionally) averaged over 500 runs, and apply smoothing to the curves.

4.2 Results

Performing the experiment we get results, depicted in Figure 3. The first thing we may observe is that all the curves with $c \leq 0.25$ perform almost equally well. This is because, regardless of c, firstly algorithms investigate all actions, because there is 0 in denominator ($n(b)$). However, low boundary options, as 0.01 and 0.05 perform slightly worse at the end. This is likely because they did not explore enough and in some of repetitions chose wrong best action because of different reward probabilities. Secondly, for bigger values, we may see from the graph, that the exploration parameter c (for big enough values) has a relation, which is close to linear, to the shape of the curve: we change the parameter exponentially (e.g. $0.5 * 2 = 1.0$), and the distances between the curves at the beginning change exponentially as well (curve of 0.5 being almost in the middle of 0.01 and 1.0). The explanation behind it may be that all these values are relatively small part of the maximum average reward possible - 1.0. When we take significantly large values, like $c = 1.0$, we put too much onto the exploration. It may work, if we increase the number of the time steps, similarly as the curve $c = 0.5$ converges to the curves with the lower exploration values. However, the options with the smaller c get closer to the max value faster, so they are the preferable option anyway.

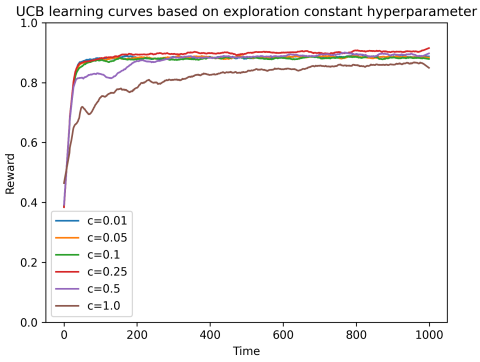

Figure 3.1: Corresponding learning curves (each point is immediate reward averaged over repetitions).

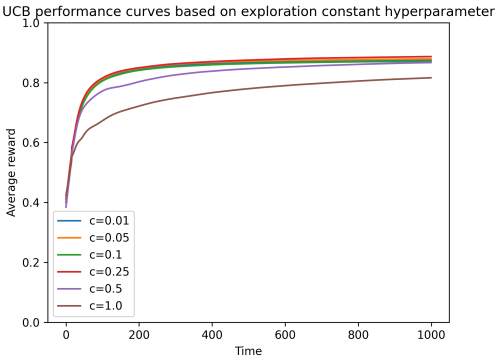

Figure 3.2: Corresponding performance curves (each point is average reward of the run before time step, averaged over repetitions).

Figure 3: Effect of exploration hyperparameter c from UCB policy formula on its performance over 1000 time steps. Reward (left) and average reward (right) at each time step is averaged over 500 repetitions, curves are smoothed with smoothing window equal to 31. The number of possible actions is 10. Among the 6 considered assigned values, we can see a clearly weaker performance of $c = 1.0$. Other settings give very similar overlapping curves, with $c = 0.5$ learning slower at the beginning (left), thus having lower average reward (right). $c = 0.25$ gives the best curve.

5 Comparison

5.1 Methodology

To compare the 3 algorithms analysed above 2 methods will be used: performance over all considered values of each hyperparameter, and over the best one. For the first method, the measurement of performance will be average total reward ($\bar{r}$), which is average reward of the whole run, on average for repetitions. Or simpler:

$$\bar{r} = \frac{1}{N \cdot T} \sum_{n=1}^{N} \sum_{t=1}^{T} r_{t,n} \tag{6}$$

, where N is the total number of repetitions ($N \in \mathbb{N}$), T - total number of time steps ($T \in \mathbb{N}$), n and t - certain repetition's number and time step ($n, t \in \mathbb{N}$), $r_{t,n}$ - immediate reward on time step t and repetition n ($r_{t,n} \in \{0, 1\}$). An alternative measure could be reward at the last time step, averaged over 500 runs; or discounted average total reward (with preference for later time steps). Possible advantages of these are noted in Appendix 7.2.
Parameters used for each algorithm correspond to those, investigated in preceding sections: $\epsilon = \{0.01, 0.05, 0.1, 0.25\}$, init_val=$\{0.1, 0.5, 1.0, 2.0\}$, $c = \{0.01, 0.05, 0.1, 0.25, 0.5, 1.0\}$.
Second method uses the best parameter, based on measure from the first method, and analyses learning curves of ϵ-greedy, OI and UCB (with the found settings). Equivalently, performance curves could be used, but they show less distinguished results, thus we do not include their analysis here.

5.2 Results

Figure 4.1 below shows the first described method of comparing the algorithms. If we consider only the optimal parameters (the highest points of each broken line), then UCB performs best, with OI having similar performance (0.88 compared to 0.86), and ϵ-greedy being the lowest (0.83). Overall location of the broken lines also indicate such rating, and preference of UCB over OI becomes more clear overall. However, such difference might be because of the chosen parameters (Appendix 7.3). This plot also confirms the preference for intermediate values of parameters for each of the algorithms, which was noted in the previous sections. We can see that for ϵ-greedy, too little exploration parameter is a bit worse than too big. For OI, the difference is more significant, which is understandable, given algorithm's need for big enough values by design. Regarding UCB, smaller parameters

still provide sufficient results, and the real problem is if the exploration constant is too high.
Figure 4.2 draws our first observation (regarding best settings for parameters) in the previous comparison method in more detail. We can indeed see that UCB has highest total average reward (biggest area under the curve). Besides that, OI has, logically, better starting point. On this plot we can also see that, in fact, OI performs as good as UCB on the longer run, in terms of rewards on the final steps. It also becomes more clear that ϵ-greedy is a worse-performing algorithm, both because: it has more gradual increase of finding best action; and its exploitation is less later in the time steps.
To conclude, UCB is the best option of the policy, with $c = 0.25$. It explores best actions fast at the beginning, and exploits them more later. Even though OI also reaches same level of final rewards, it is slower in exploring at the beginning. A way to solve this would be to find more optimal initial value in order to lessen distance to the needed point; or use another policy, instead of greedy to decrease faster (for example, also ϵ-greedy for higher exploration parameter). Regarding ϵ-greedy, an improvement could be its another version, described in Appendix 7.1. We could also look for more optimal value (for instance, using binary search).

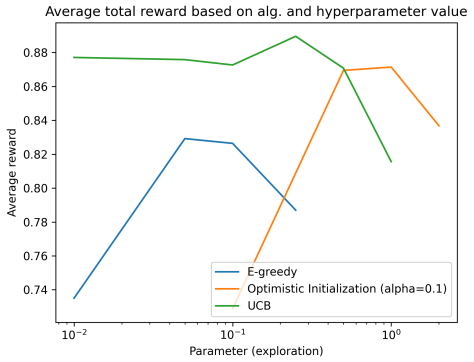

Figure 4.1: Comparison of ϵ-greedy, OI and UCB over each of their hyperparameters. Average (total) reward for each algorithm is calculated over all time steps and all runs. UCB shows best performance and ϵ-greedy the worst with their optimal parameters. However, worst performance is observed for OI policy.

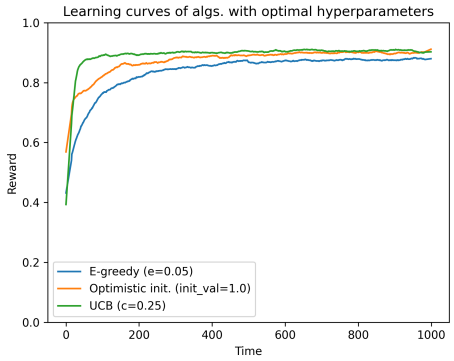

Figure 4.2: Comparison of learning curves of ϵ-greedy, OI and UCB with their best parameters (maximums on Figure 4.1). Averaged over 500 repetitions, smoothing window is 31. UCB learns fastest in the start, and OI has highest start of the curve (first reward). They outperform ϵ-greedy also in terms of exploitation in the end.

6 Conclusion

In this research, we have investigated different algorithms and their hyperparameters for 10-armed bandits problem. We have looked at different values of epsilon for ϵ-greedy policy, of initial values for OI policy, and of exploration constant for UCB policy. We have found that extreme values work worst: either because of initial learning speed, or final received best rewards. For the given formulation of the problem, the best constants were found to be: $\epsilon = 0.05$ or 0.1; init_val $= 0.5$; $c = 0.01$, 0.05, 0.1 or 0.25. Such findings are confirmed by both curves of interest - learning (where target is immediate reward) and performance (where target is average reward over preceding timesteps).
Including found results, we have compared all 3 algorithms based on their average total reward and best hyperparameters. In both cases, Upper Confidence Bound policy was found to be the best (in particular, for exploration constant equal to 0.25), because it has 1) highest average total reward, 2) fastest exploration at the beginning, and 3) optimal exploitation in the following time steps. Optimal Initialization policy performs worse in terms of slower exploration at the start, and lower broken line of average total reward for non-optimal parameters. ϵ-greedy policy was found to be worse than both other policies in all 3 named characteristics.
Weaknesses found in every section are summarized in Appendix. For further research we propose implementing these improvements in the current version of experiment (in particular, denser distribution of parameter values). We also suggest adding more algorithms (for example, Stochastic Gradient Ascend) or problem setup settings for more global comparison (for example, k-armed bandits for different k).

REFERENCES

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.