Learning Dynamics / Computational Game Theory Assignment 3: Reinforced Learning

Horatiu Luci
Year 1 - M-SECU-C @ ULB
VUB Student ID: 0582214
ULB Student ID: 000516512

22 December 2020

Exercise 1: N-Armed Bandit

For this exercise, 3 classes have been created: Softmax, Egreed and Random. Each class contains a pull method, based on which it will select the next action as follows:

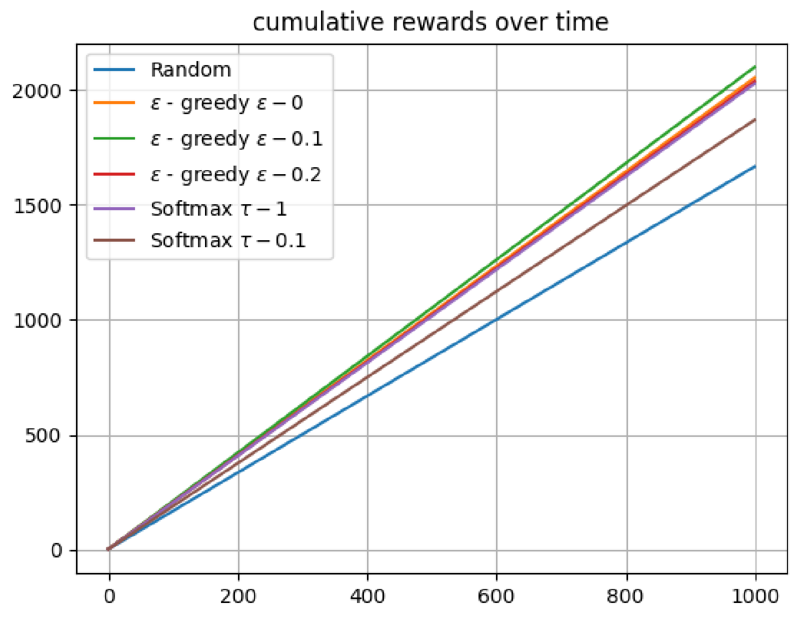
- $\epsilon-$*greedy*: A random action will be chosen with probability ϵ and the best known action will be selected otherwise. Essentially if ϵ is 0 this algorithm will behave randomly.
- **Softmax**: The following probability distribution of choosing each arm at each given round will be used: P (explore arm i) = $\frac{e^{r_i/\tau}}{\sum e^{r_i/\tau}}$. Essentially, when τ has a high value, the chance of exploring any arm is equal.
- **Random**: An action will be chosen at random using python's randint function.

Based on my ULB enrolment number (000516512) ending with 2, Table 3 was chosen and it looks as follows:

Action	Q^*_{ai}	σ_i
action #1	2.2	0.4
action #2	2.6	0.8
action #3	1.0	2.0
action #4	1.4	0.6

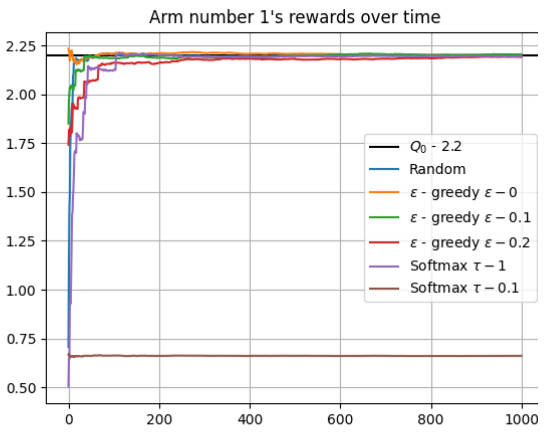
Exercise 1.1

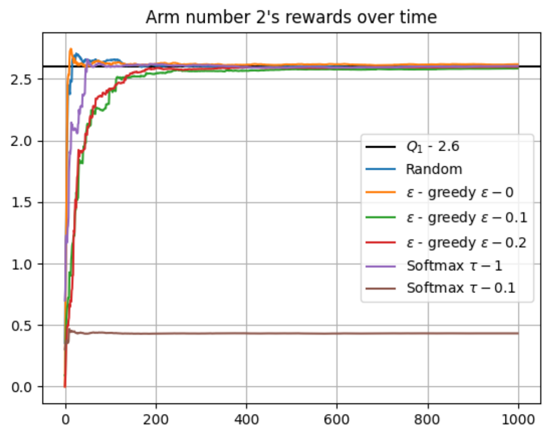
Cumulative averaged reward over time

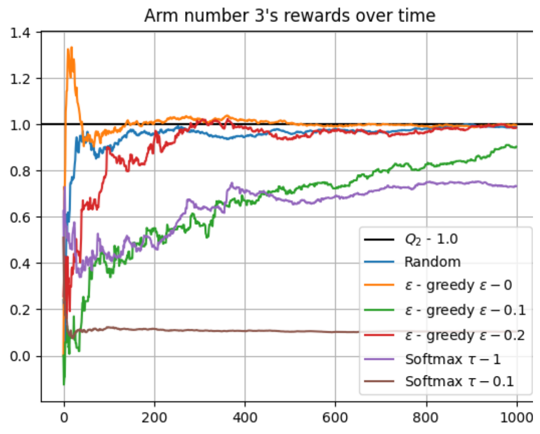


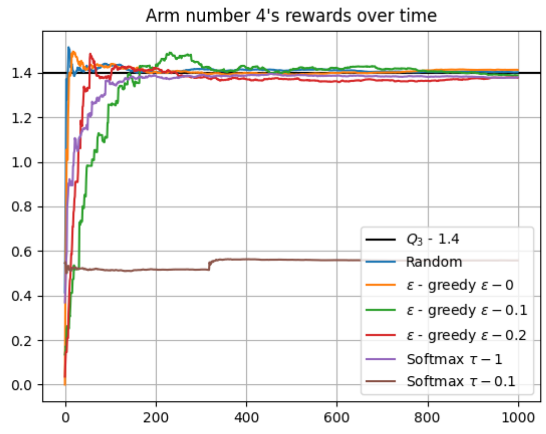
After 1000 runs averaged over 30 simulations, the cumulative rewards indicate that $\epsilon - greedy$ **algorithm with parameter** $\epsilon = $ **0.1 is the algorithm that gathered the most reward**, also with $\epsilon = 0$ or 0.2 as well as Softmax with parameter $\tau = 1$ managing to stay close to the top earnings. However, Softmax with a small parameter behaves poorly as well as the random algorithm as expected.

Individual arm plots



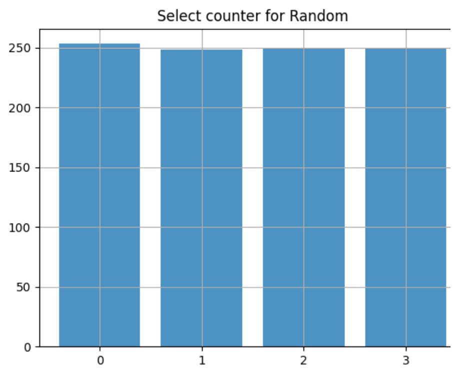


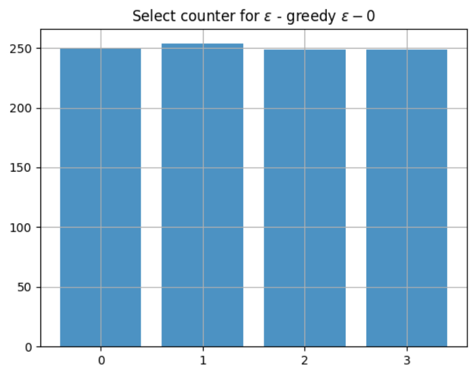


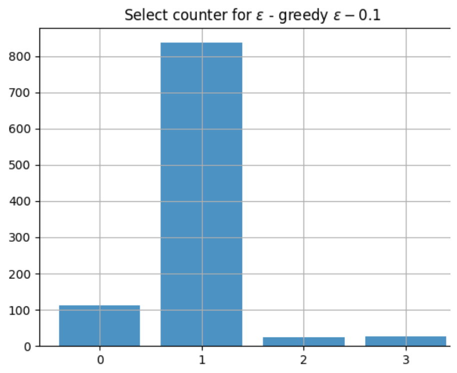


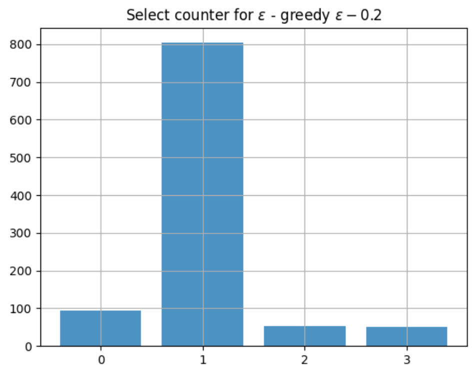
Here it can be interestingly seen that as the third arm offers the lowest reward mean but has the biggest deviation, especially greedy algorithms will avoid it if the initial rewards are small. However, if the initial rewards are big, the greedy algorithms will convey towards it, revealing rapidly that the rewards are not actually that big.

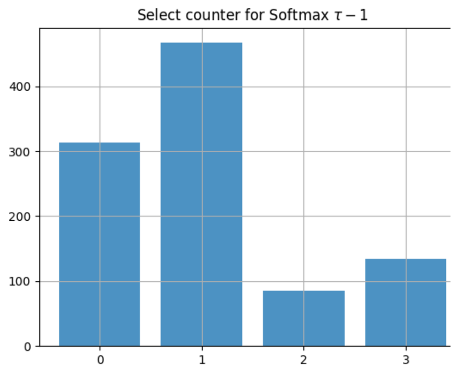
Selections made

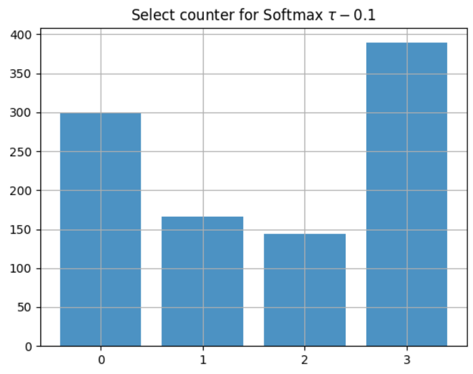










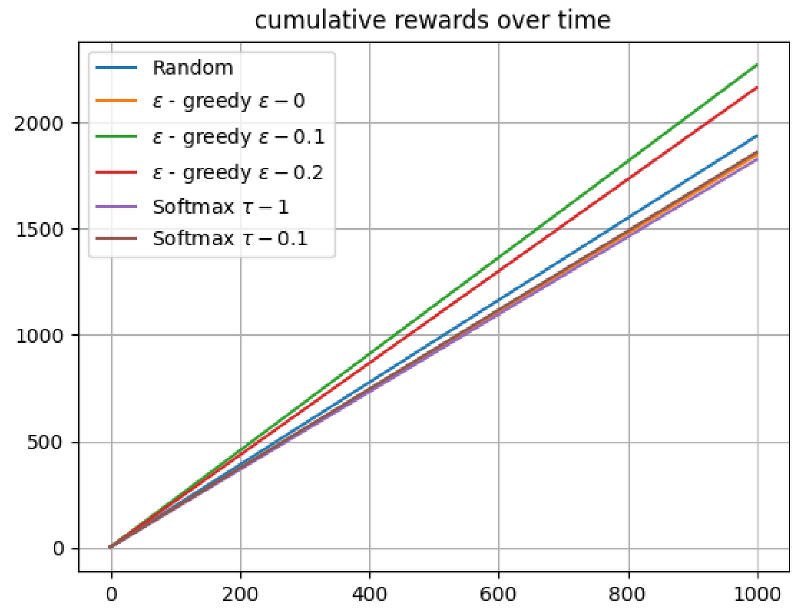


As expected, then ϵ is 0, the algorithm behaves randomly. This random behaviour can be seen as well for small values of τ in the Softmax algorithm. For especially greedy algorithms, the second arm was chosen the most as it has the best reward. For Softmax with big values of τ, the arms are chosen with a frequency directly proportional to the respective arm's reward.

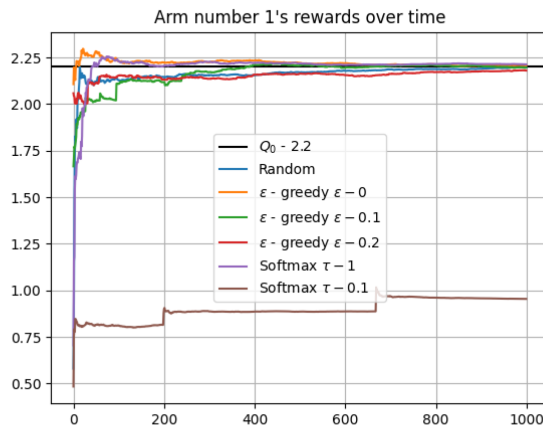
Exercise 1.2

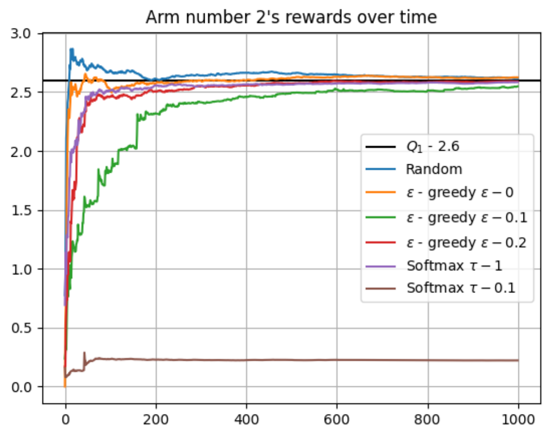
For this exercise, the same was run as before, except the standard deviations for each action have been doubled. Due to this fact, the algorithms were having more difficulty identifying the optimal arm. However, due to the same fact, more rewards were gained over time. Best performer is still greedy with $\epsilon = 0.1$ The plots are below.

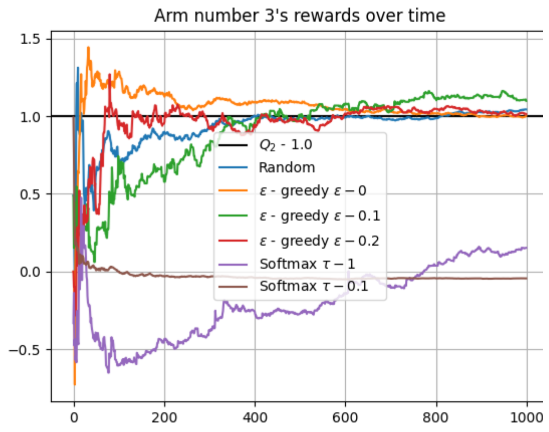
Cumulative averaged reward over time

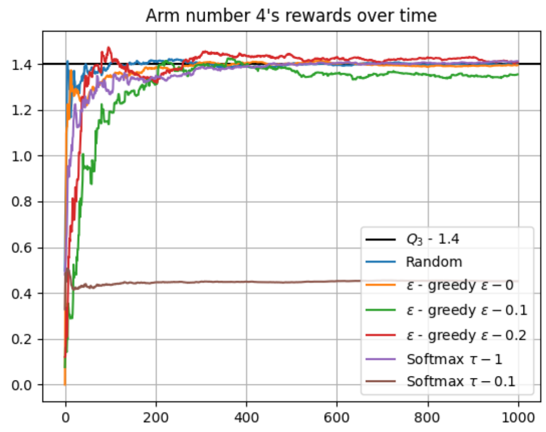


Individual arm plots

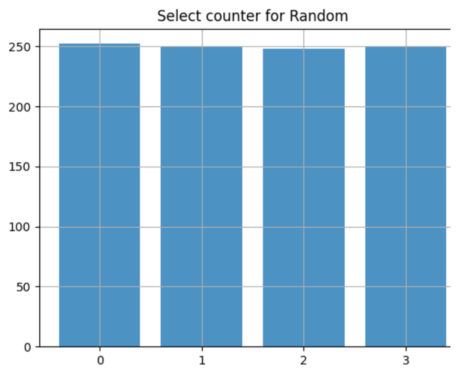


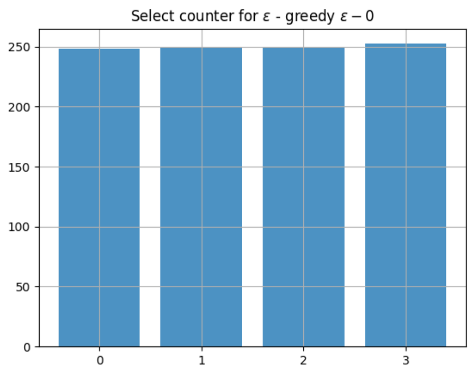


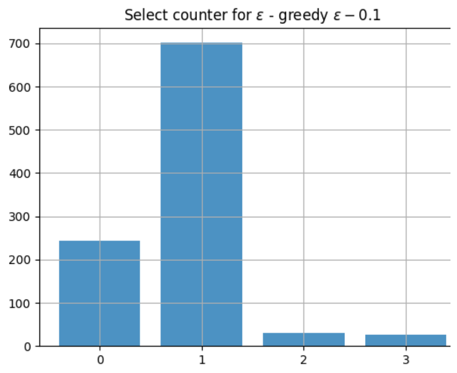


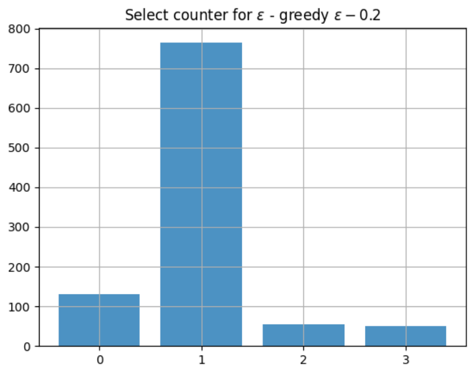


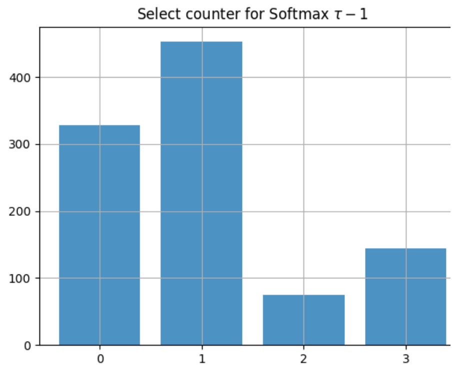
Selections made

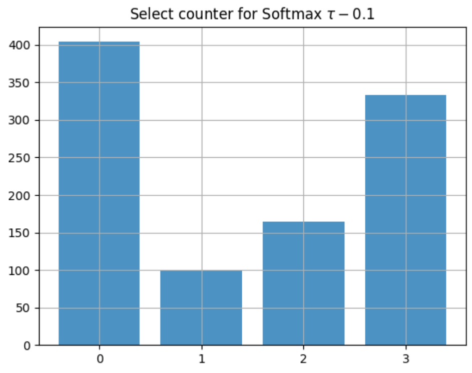








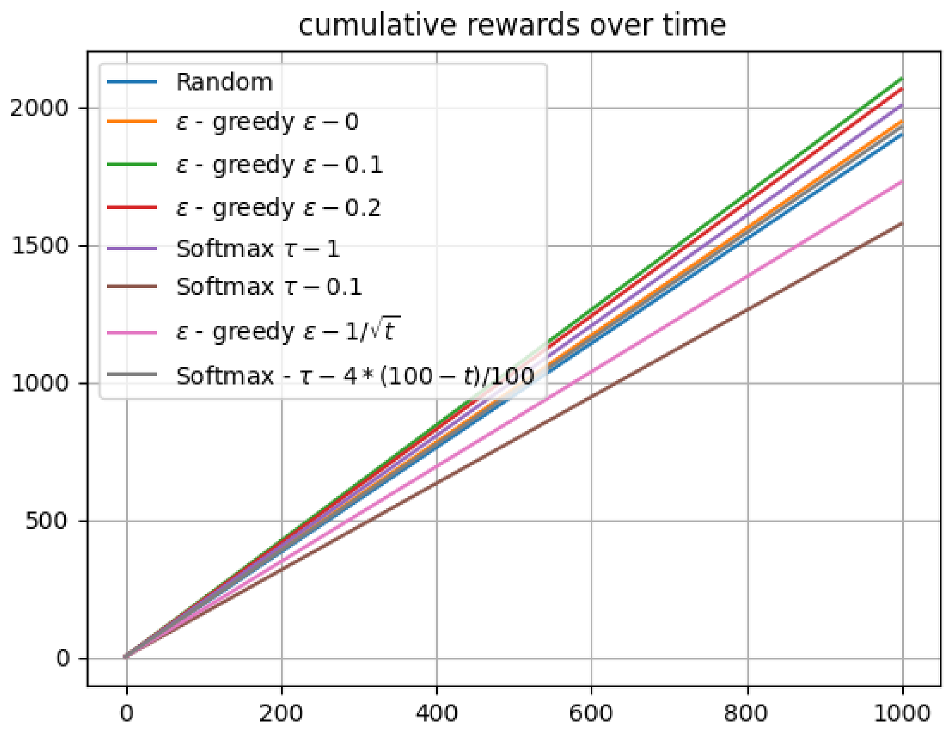




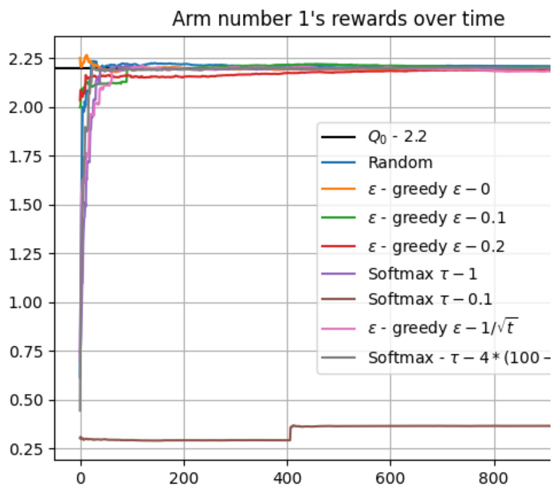
Exercise 1.3

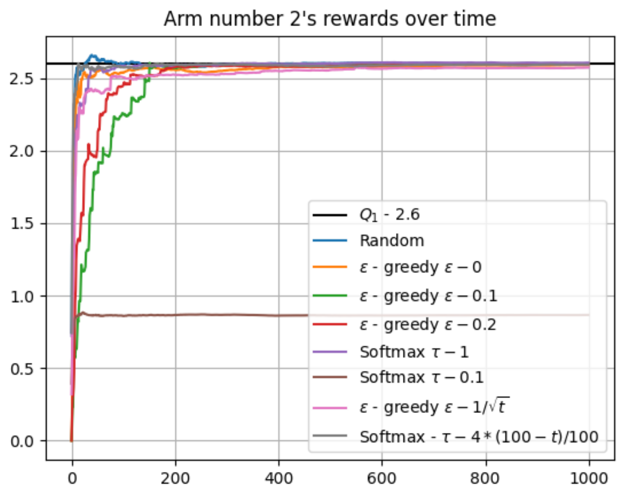
As it can be seen from the below plots, the algorithms run worse when including a time parameter into their hyper-parameter. For $\epsilon - greedy$ the algorithm becomes more and more random as time goes on, starting from 1, essentially the opposite of the ideal.
However, for Softmax, this parameter works a bit better than the previously used ones, as the parameter is decreasing from 4 to 0 as rounds go by, the algorithm becomes more and more biased towards the known big rewards. The plots are below:

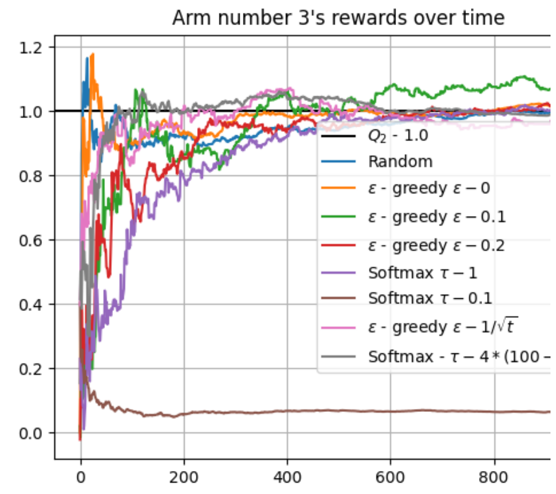
Cumulative averaged reward over time

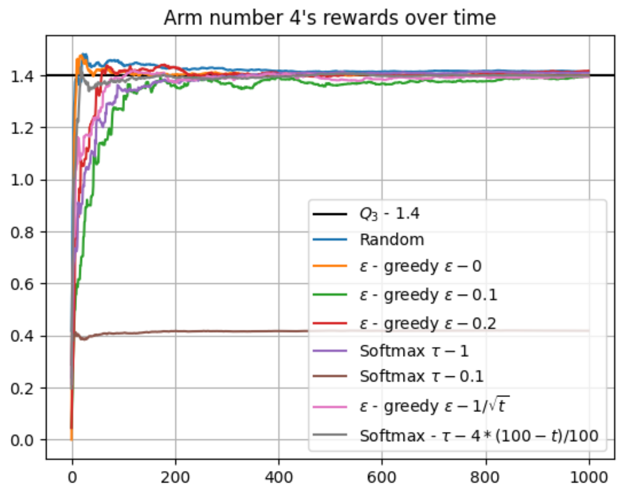


Individual arm plots

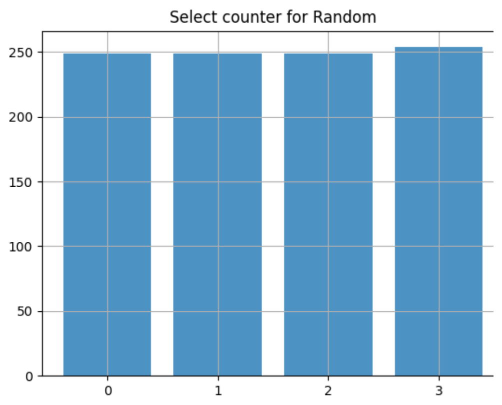


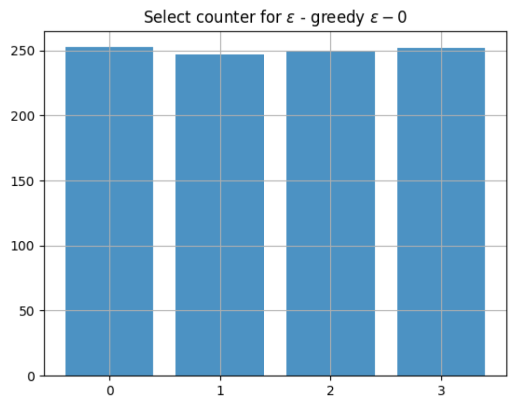


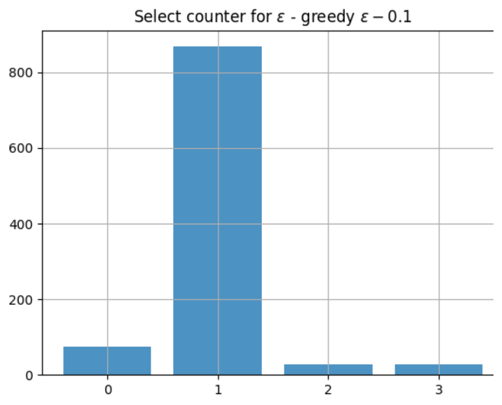


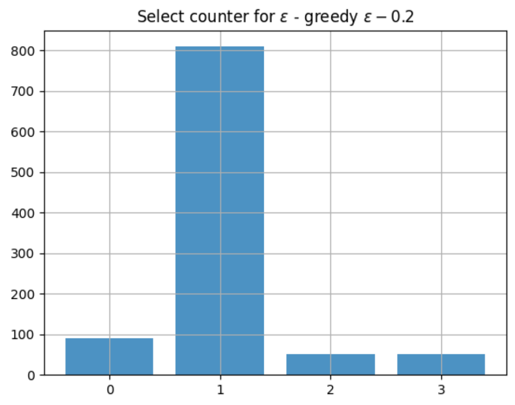


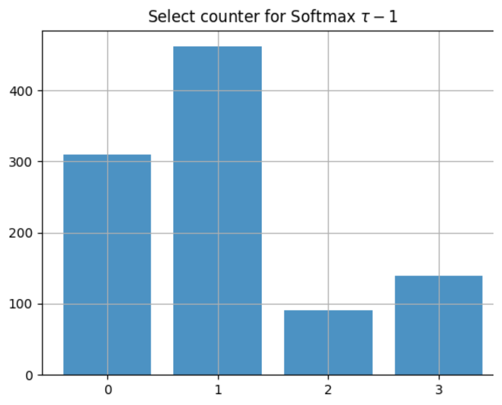
Selections made

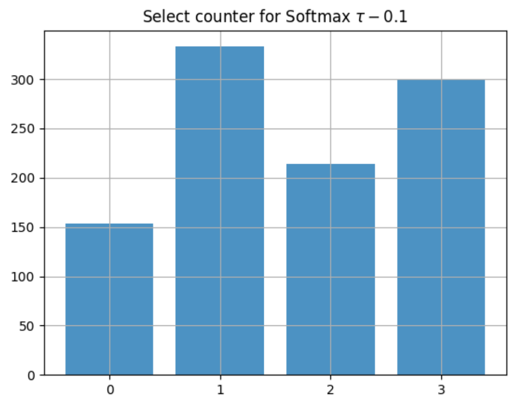


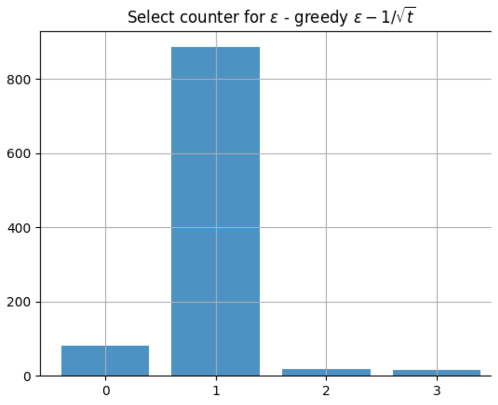


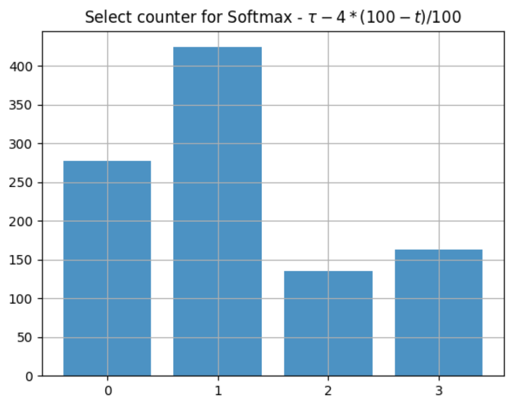









Select counter for ε - greedy $\varepsilon - 1/\sqrt{t}$
0
200
400
600
800
0
1
2
3


Select counter for Softmax - $\tau - 4 * (100 - t)/100$
0
50
100
150
200
250
300
350
400
0
1
2
3

References

[1] https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-epsilon-greedy-algorithm-8057d7087423

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

[3] https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/

[4] https://www.datahubbs.com/multi-armed-bandits-reinforcement-learning-2/