# Reinforcement Learning Practical, Assignment 1

Mohamed Gamil(s3897605), Sofia Teeriaho(s3887626)

December 7, 2021

## Algorithm descriptions

The algorithms described below are used to navigate an agent's actions when facing a simulated multi-armed bandit problem. In a basic sense they attempt to achieve a balance between exploration and exploitation to achieve the maximum reward received.

### Greedy

As the name suggests, the greedy algorithm is defined as letting the agent take the action that has the highest estimated value and exploit it without further exploration. However, it is very likely that the arm/action being exploited is not the best. Thus, the reward received is not maximised.

Formula for Action-Value estimate:

$$\begin{aligned} Q_t(a) &\doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} \\ &= \frac{\sum_{i=0}^{t-1} R_i \cdot I(A_i = a)}{\sum_{i=0}^{t-1} I(A_i = a)} \end{aligned}$$

Mathematical formula of the greedy method:

$$A_t = \underset{a \in \mathcal{A}}{argmax}\ Q_t(a)$$

### Epsilon-greedy

The epsilon-greedy approach aims to solve the problem that Greedy has by giving a probability of random exploration at each action. A value (epsilon) is pre-set before the experiment, such that $0 <$ epsilon $< 1$. This gives the agent a slight chance to explore other actions(probability = e) while exploiting the current best action(probability = 1-e). Obviously, this gives better results than greedy.

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

### Optimistic initial values

This algorithm works by first assuming that all actions yield rewards larger than reality. Since the estimations are unrealistically large, the agent notices the difference and picks a different action the next time all while lowering the estimation for that action. Process is repeated until the estimated reward converges to the true value.

**Softmax Policy**

The Softmax policy operates by creating a probability for each action to be picked that is proportional to the average reward it provides.

The mathematical formula:

$$\pi(a) = \frac{e^{\frac{r}{\tau}}}{\sum_{k=0}^{N} e^{\frac{r_k}{\tau}}}$$

where r is the current average reward for action a and $\tau$ is a hyper parameter used to control the randomisation.

**Upper-Confidence Bound**

This strategy directs the agent's exploration based on the uncertainties in the actions. The uncertainties depend on how many times an action is selected so far.
The mathematical formula:

$$a_t \doteq argmax_{a \in \mathcal{A}} \left[ Q_t(a) + c\sqrt{\frac{ln(t)}{N_a(t)}} \right]$$

$N_a(t)$ represents how many times action a was picked.
c is a controlling parameter used for adjusting the exploration amount.
The main idea is that the agent will always choose the action with the highest UCB. Thus, actions that are least explored(high uncertainty ) will be chosen.

**Action Preferences**

This algorithm sets preferences of actions to be chosen. It updates the preferences after the selections using the two rules shown below.

$$H_{t+1}(a') \doteq H_t(a') + \alpha\,(r_t - \bar{r}_t)(1 - \pi_t(a'))$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\,(r_t - \bar{r}_t)\,\pi_t(a) \;\text{ if }\; a \neq a'$$

Where a' is the selected action, reward is $r_t$, $\pi_t(a)$ is the selection probability , and $\bar{r}_t$ is the average reward.
Unfortunately we were unable to present the results of this algorithm in this multi-armed bandit simulation.

## Experimental Setup

For Gaussian distribution the true reward value for each experiment (N) is defined with the `numpy.random.randn()` function. The reward received each time step (T) per single experiment is drwan by using the numpy function `numpy.random.normal(self.true, self.std)`. This function takes into consideration a standard deviation of 1 and the true reward value for that specific action. For the bernoulli distribution the function `generate rewards` is used to first generate an array of true reward values, in this case the probability of drawing the binary reward 1. Once we have the probability for each action, we can generate an array of ones and zeros for each time step and store these in the action class. This is done using the function `rewards = np.random.rand(T, K) < np.tile(p, (T, 1))`. For every experiment the time step is used as an index to retrieve the reward value from the generated array of rewards for each action.
For bernoulli, we chose the following parameters: 5 actions, 100 time steps and 500 experiments. We noticed that when an agent has more actions to choose from, for example 10 actions instead of 5 then the average reward value for each time step varies more resulting in larger oscillations in the plot. For

epsilon-greedy, an epsilon value of 0.1 was chosen and for optimistic initial values the starting reward value estimate was 1 given that this is the only optimistic value in bernoulli distribution.
For gaussian, we chose the following parameters: 10 actions, 100 time steps and 300 experiments. For epsilon-greedy, an epsilon value of 0.2 was chosen to slightly avoid the convergence with the softmax method. For the starting reward value estimate for optimistic initial values we chose 3.

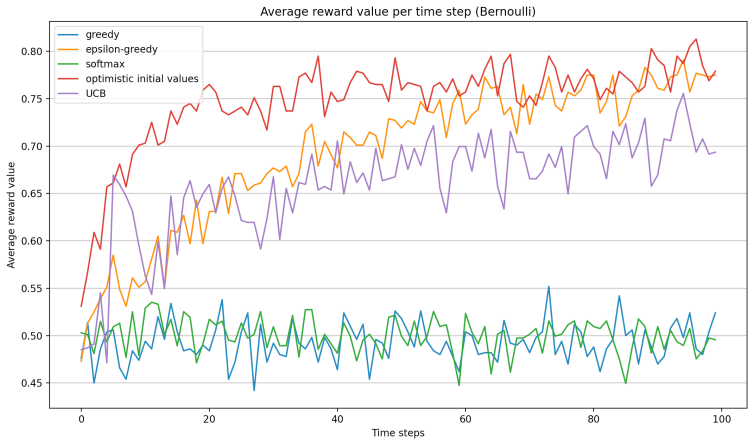
## Results

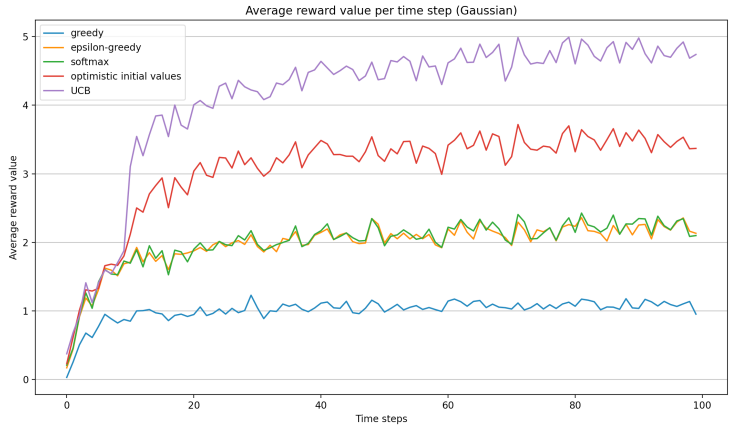
The following values in the table were obtained by finding the amount of times the agent chose the best action per the number of total actions taken in that experiment. The average value over all experiments is what is plotted below.
Average Percentage of times the best action is selected over all experiments (Bernoulli)

| | Greedy | E-Greedy | Softmax | OIV | UCB |
|---|---|---|---|---|---|
| **Percentages:** | 19.8 | 48.824 | 20.2 | 62.208 | 41.488 |

Average Percentage of times the best action is selected over all experiments (Gaussian Distribution )

| | Greedy | E-Greedy | Softmax | OIV | UCB |
|---|---|---|---|---|---|
| **Percentages:** | 31.73 | 45.963 | 9.667 | 50.467 | 55.953 |




## Analysis

As expected, the greedy algorithm tends to converge earlier on a lower average reward, since it goes straight into exploiting one arm without further exploration. The epsilon-greedy algorithm performs better compared to Greedy. Which could be explained with the concept that a pre-set exploration

parameter, which motivates the agent to inform itself about other actions improves the chances of the agent finding the best action

Given from the percentages tables ,we could see that the Softmax policy algorithm could not find the best action in both problems, which did have an effect on the average reward value obtained as seen in the plots. This means that it was not able to exploit the best arm. Perhaps, if it had been used with the action-preferences update mechanism described above, the results would have been improved.

We see that the UCB algorithm performs better than the rest on the Average reward plot in the Gaussian distribution problem. This is due to the fact that the agent learns to change its beliefs about which arm will result in higher rewards given that the agent is exposed to the true payoff each time step and therefore uncertainty is reduced and the agent can adjust its beliefs accordingly. Also, it seems to always achieve the highest percentage of choosing the best action in the gaussian-bandit.

The Optimistic Initial Value algorithm performs well in relation to the rest of the algorithms -especially in the Bernoulli problem- since it is set to explore at an early stage which allows the action-value estimates to approach the true reward values faster. It then exploits the best action found which allows the average reward to be maximised. Further evidence that it is quicker with finding and picking the best action is the percentages shown in the table above for both situations.