INTRODUCTION TO RL ASSIGNMENT 1: BANDITS

Bas Blokzijl (s2605457) and Enrico Bonsu (s3045072)

Universiteit Leiden
Introduction to Reinforcement Learning 2022-2023

1 INTRODUCTION

In this report, we will look at the effects of three different evaluation aspects of reinforcement learning that are applied to a trivial problem. The problem is known as the k-armed Bandit Problem. In this problem at each time step t one out of k actions can be selected. The goal is to maximize the possible reward that can be received.
To solve this problem information gathered after interacting with each arm must be used in order to improve the decision made in the future. This process can be divided into three parts;

1. Initialization of the knowledge of the agent about the environment.
2. Selecting an action based on this knowledge.
3. Update the knowledge of the agent after executing the action.

The problem arises from the uncertainty of the reward that an action a will generate, which the agent does not know beforehand. Consequently, the agent must rely on its experience of receiving a reward r after performing the action a. The evaluation algorithms that we will discuss in this report make use of an estimate for the reward received after executing action a. However using an estimate is not the only way to use the information it gains after performing an action. In Appendix A we will look at the Gradient Bandit algorithm which uses numerical preference instead of an estimate reward to select an action. We start with looking at how the ϵ-greedy algorithm perform in the k-armed bandit problem, followed by the Optimistic Initialisation value algorithm and lastly the Upper-Confidence-Bound Algorithm. In the last section these algorithms are compared to each other.

1.1 THE EXPERIMENT

The environment in which we set up our experiments have 10 actions available at each time step. Each run completes 1000 time steps. To gather a consistent result we average over 500 runs. After selecting an action a reward of either 0 or 1 is returned. The mean pay-off for each action is known to the environment from the start, but not to the agent. The mean pay-off is also stationary for each action in the environment. The goal for the agent is to receive the highest possible reward as soon as possible. Let us first look how the ϵ-greedy algorithm performs in this environment.

Definition Bandit
A bandit is defined by the tuple:

$$\{\mathbb{A}, p(r|a)\} \tag{1}$$

Where A is a set of actions.
$p(r|a)$ *is a conditional probability distribution, mapping each action to a distribution over the possible rewards*

2 ϵ-GREEDY ALGORITHM

In this section, we will look at the methodology of the ϵ-greedy algorithm and the results it produces for the k-armed bandit problem. This algorithm is a modification of the greedy algorithm.

2.1 METHODOLOGY

The ϵ-greedy algorithm is a modification of a greedy algorithm. A greedy algorithm is characterised by picking the best action at each stage, however many problems are stochastic in nature. If the problem is stochastic the agent picks an action based on the expected reward it can received from executing action a. To track this expected value received from executing action a the agent stores an estimate of the rewards it has received over the number of times the value is picked. The only difference between the ϵ-greedy algorithm and the greedy algorithm is that the ϵ-greedy algorithm does not always pick the action that is estimated to produce in the highest reward. The hyper parameter ϵ in the ϵ-greedy algorithm introduces a core concept in reinforcement learning called exploration and exploitation. Exploration refers to selecting an action that is not based on its (estimated) reward, while exploitation refers to selecting an action that is based on its (estimated) reward. For the ϵ-greedy algorithm the value for ϵ is the probability that a random action is selected. To generate a greedy algorithm from a ϵ-greedy algorithm the ϵ value must be set 0. In that case the algorithm would always picks the action that is estimated to result in the highest reward.
The implementation of the ϵ-greedy algorithm can be found in Algorithm 1 which were provided by the lecture notes (1).

Definition Greedy Policy
Given a set of actions $\mathbb{A}$ and action value $\mathbb{Q}$ we define the greedy policy as:

$$\pi(a) : \mathbb{A} \to \{0, 1\} : a \mapsto \begin{cases} 1, & \text{if } a = \arg\max_{b \in \mathbb{A}} Q(b) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Algorithm 1 ϵ-greedy algorithm.

procedure
  Input: Exploration parameter $\epsilon \in [0, 1]$, maximum number of timesteps T.
  Initialization: Initialize $Q(a) = 0$, $n(a) = 0\ \forall a \in \mathbb{A}$
  for $t = 1 \dots T$ **do**
    $a_t = \begin{cases} \arg\max_{a \in \mathbb{A}} Q(a), & \text{with } p = 1 - \epsilon \\ \text{random non-greedy action}, & \text{with } p = \epsilon \end{cases}$
    $r_t \sim p(r|a_t)$
    $n(a_t) \leftarrow n(a_t) + 1$
    $Q(a_t) \leftarrow Q(a_t) + \frac{1}{n(a_t)} \cdot [r_t - Q(a_t)]$
  end for
end procedure

Let us now compare the results of the ϵ-greedy algorithm on the k-armed Bandit Problem for different hyper-parameter values for ϵ.

2.2 EXPERIMENT RESULT

Looking at Figure 1 we observe the effect of different ϵ values using the ϵ-greedy algorithm. We see that for $\epsilon = 0.01$ the algorithm uses many more time steps to reach a good result while having $\epsilon = 0.25$ the algorithm converges to the lowest reward out of the four ϵ values tested. This tells us that finding a balanced value for ϵ is essential for the problem. We do not want to have ϵ too low which would result in the agent only so often trying to explore other actions while having ϵ too high results in the agent exploring too much. Clearly, the figure shows the trade-off (for the different values for ϵ) between exploration and exploitation. For this problem we yield the best result when using an ϵ value of 0.1

3 OPTIMISTIC INITIALISATION ALGORITHM

This section provides a detailed account of our implementation of the Optimistic Initialisation algorithm, focusing on the theory that underpins the algorithm as well as the equations utilized. We conclude this section by providing the implementation of this algorithm.

Figure 1: ϵ-greedy algorithm average performance over 500 repetitions for different ϵ values.

3.1 Methodology

The main idea behind the Optimistic Initialisation (OI) algorithm is to use the initial value functions as an incentive for exploration. The latter is achieved by initializing the value estimate of all actions with a (overestimated) high positive value. This means, that whichever action is first selected, its estimated reward value will become significantly lower than that of the unexplored actions. In turn, leading to the algorithm exploring other possibilities. After sufficient iterations the value estimates will converge but only after each action was tried several times. Note that the larger the number of possible actions and the larger the reward set at initialisation, the longer it takes for the value estimate to converge and more exploration will be done.
The implementation of the Optimistic Initialisation algorithm can be found in Algorithm 2 which were provided by the lecture notes (1).

Algorithm 2 Optimistic Initialisation (with greedy actions selection).

procedure
  Input: initial value $\psi \in \mathbb{R}$, learning rate $\eta \in \mathbb{R}^+$, maximum number of time steps T.
  Initialisation: Initialize $Q(a) = \psi \; \forall a \in \mathbb{A}$
  for $t = 1 \dots T$ **do**
    $a_t = \arg\max_{a \in \mathbb{A}} Q(a)$
    $r_t \sim p(r|a_t)$
    $Q(a_t) = Q(a_t) + \eta \cdot [r_t - Q(a_t)]$
  end for
end procedure

3.2 Experiment result

During our experiment we fixed our learning rate η to 0.1 for each of the initial values tried.
We see a clear drop in performance for the smallest initial value, it is also obvious that the two highest initial values defeat the lower ones in the long run. This is not surprising. In essence the OI algorithm is a greedy algorithm. When we set our initial value close to the worst possible outcome (in this case 0 is the lowest reward possible) the algorithm very early sticks with the action whose estimated reward is greater than 0.1. Since the mean pay-off of the actions in the environment are

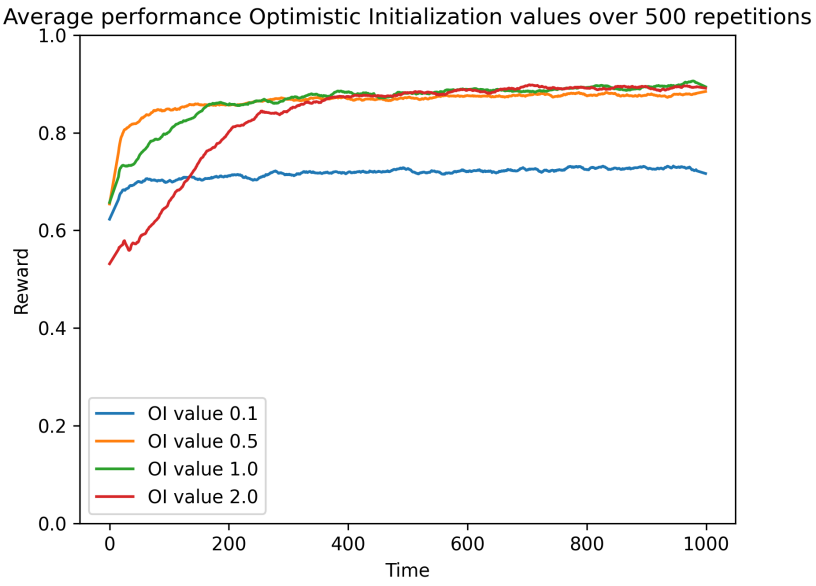

Figure 2: Optimistic Initialisation algorithm average performance for different initial values.

between 0.0 and 1.0 it is easy to find an action that does not yield the best mean pay-off reward but is higher than the initial estimated value (of 0.1). The would result in the algorithm to stop exploring and keep on exploiting the found action. The second worst performing initial value is 2.0, this is caused by the fact that the mean pay-off of the actions can be at most 1.0. Having all estimated reward values over this maximum mean pay-off imply that all the actions are actively exploited, and continues to be re-estimated until they come close to their actual mean pay-off. This requires many time steps to recover the actual reward value of the actions.

4 Upper-Confidence-Bound algorithm

This section explains the theory and gives a practical implementation for the Upper-Confidence-Bound algorithm (UCB).

4.1 Methodology

UCB algorithm is another algorithm to select actions that balance exploration and exploitation. The idea behind UCB is to use uncertainty to drive exploration, by selecting actions that have buildup uncertainty because they have not been picked recently. It works by maintaining an estimate of the expected reward for each action, based on the agent's past experience, and increase (or decrease) the potential to be picked by using uncertainty of the result of the action. The estimate is typically a running average of the rewards received for each action. UCB takes advantage of this uncertainty to choose actions that have the potential to be highly rewarding.
The implementation of the Upper-Confidence-Bound algorithm can be found in Algorithm 3 which were provided by the lecture notes (1).

Defition UCB Policy
Given a set of actions $\mathbb{A}$, exploration parameter $c \in \mathbb{R}^+$ and action value $\mathbb{Q}$ we define the greedy policy as:

$$\pi_{\text{UCB}}(a) = f(Q, n, c) : \mathbb{A}\times\mathbb{R}\times\mathbb{R}^+ \to \{0, 1\} : a \mapsto \begin{cases} 1, & \text{if } \ a = \arg\max_a \left[Q(b) + c \cdot \sqrt{\frac{\ln(t)}{n(b)}}\right] \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Algorithm 3 Upper-Confidence-Bound algorithm.

procedure
  Input: Exploration parameter $c \in \mathbb{R}^+$, maximum number of timesteps T.
  Initialisation: Initialize $Q(a) = 0$, $n(a) = 0\ \forall a \in \mathbb{A}$
  for $t = 1 \dots T$ **do**
    $a_t = \arg\max_a \left[Q(b) + c \cdot \sqrt{\frac{\ln(t)}{n(b)}}\right]$
    $r_t \sim p(r|a_t)$
    $n(a_t) \leftarrow n(a_t) + 1$
    $Q(a_t) \leftarrow Q(a_t) + \frac{1}{n(a_t)} \cdot [r_t - Q(a_t)]$
  end for
end procedure

4.2 RESULTS

We ran the UCB agent with several different *confindence* values, the performance can be found in Figure 3.
The UCB agent's performance seems to improve linearly when the confidence parameter is set to higher values for the first few values set. We expected this to happen; for lower confidence values the agent does not explore enough and the performance diminishes. So, iterations with a higher confidence value will likely yield better results. This is clearly reflected by the model converging quickly to a high reward value. For the last two confidence values used, we see the performance decline again. It takes longer for the model to converge and when it does the reward is lower than that of other versions. We think this is due to the agent exploring excessively, the agent becomes too reckless and overly weighs the exploration term in the selection of actions.

4.2.1 CONCLUSION

Based on our analysis we conclude that the UCB algorithm is sensitive to the choice of parameters, and finding the optimal values requires some experimentation and tuning. We find that it is crucial to assess the performance across a range of parameter values to find the optimal setting for the specified task at hand

5 HYPERPARAMETER COMPARISON

For this experiment, we ran all algorithms with different hyperparameter values. We see that for most hyperparameter settings UCB is the clear winner in average reward here. It is evident that for this particular problem, the difference between the expected reward obtained by the UCB agent and the best possible expected reward is very small because the amount of possible actions is low and hence uncertainty for each action is lower as well. We expected the UCB algorithm to do best as its policy gives a really good approximation for the best balance between exploration and exploitation which enables faster convergence than OI or ϵ-greedy. It is also important to note that the OI agent outperforms the best version of ϵ-greedy for 3 of its hyperparameter settings. Our prediction is that the OI agent can achieve good performance in the early stages of the experiment as exploration is highly encouraged by the initialisation values. This, in turn, leads to faster convergence hence beating ϵ-greedy. In addition, since ϵ-greedy does not converge in the same way; it will still

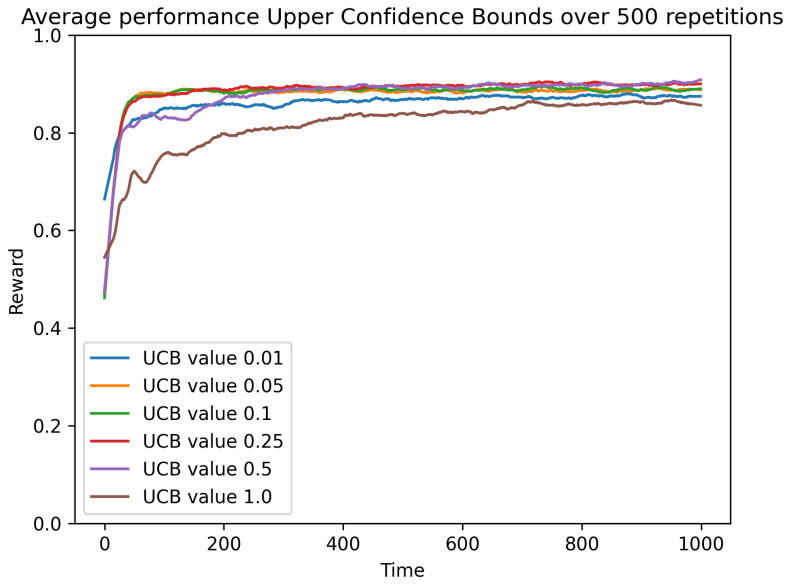

Figure 3: UCB algorithm average performance for different *confindence* values.

select actions with inferior reward estimates even when 400+ iterations in. We expect ϵ-greedy to perform worse as less favourable actions are still randomly selected even if the reward estimates are reinforced well enough to the point of convergence.

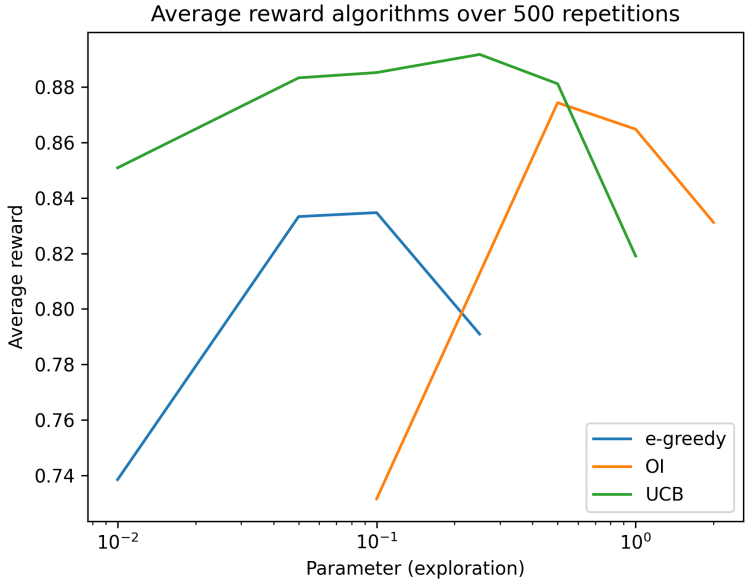

Figure 4: Comparision of all algorithms with different hyperparameters.

5.0.1 CONCLUSION

Our data shows that with the right hyperparameter settings, UCB is the best algorithm to use for this particular problem. It is clear that iteratively increasing the value of the hyperparameters does not always yield better results. For each parameter, there is a certain ”sweet spot” to be determined that correctly balances the agent’s exploitation and exploration of the given environment.

6 BIBLIOGRAPHY

REFERENCES

[1] *Lecture Notes: Bandit by Thomas Moerland, Course: Reinforcement Learning 2022-2023, Bachelor AI, Leiden University*

[2] Sutton, R. S., Barto, A. G. (2020). Reinforcement Learning: An Introduction 2nd edition. The MIT Press.

A GRADIENT BANDIT ALGORITHM

The three algorithms discussed in this report all make use of an estimate reward value for each action a (at time step t). This however is not the only way to decide which action to take. In this section we will look at the Gradient Bandit algorithm, which makes use of a numerical preference for each action a.

A.1 METHODOLOGY

The idea behind the Gradient Bandit algorithm is that based on the numerical preference of action a it can be decided whether to select this action or not. The preference for an action a, denoted by $H_t(a) \in \mathbb{R}$ is independent from the reward value received from executing action a.
The preference for an action is either increased or decreased after receiving the reward for that action. Based on the baseline reward ($\bar{R}$) the algorithm decides whether the reward received is performing better or worse than this baseline reward. The baseline $\bar{R}$ is calculated by taking the average of the rewards received up to the point of receiving the latest reward. When the reward received is higher than the baseline reward the preference for that taken action increases, while the preference for all other action decreases. In the case that the reward is worse than the baseline the perference for that action is decrease, while the preference for all other actions are increased. The implementation of the Gradient Bandit algorithm can be found in Algorithm 4 which is derived from the textbook (2).

Algorithm 4 Gradient Bandit algorithm.

procedure
  Input: Learning parameter $\alpha \in \mathbb{R}^+$, maximum number of timesteps T.
  Initialisation: Initialize $H(a) = 0 \; \forall a \in \mathbb{A}$ and $\bar{R} = 0, \bar{R} \in \mathbb{R}$
  for $t = 1 \dots T$ **do**
    $a_t = \arg\max_a H_t(a)$
    $r_t \sim p(r|a_t)$
    $H(a_t) \leftarrow H(a_t) + \alpha(r_t - \bar{R}_t)(1 - \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}})$
    for $a \neq a_t$ **do**
      $H(a) \leftarrow H(a) - \alpha(r_t - \bar{R}_t)(\frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}})$
    end for
    $\bar{R} \leftarrow \frac{1}{t} \sum_t r_t$
  end for
end procedure

A.2 EXPERIMENT RESULT

Looking at Figure 5 we observe the effect of different α values using in the Gradient Bandit algorithm. We observe that the algorithm converges quite fast for all the learning rates tried. We also see that the learning rate of 1.0 performs the worst out of all the rates tested. This does not come as a surprise because setting the learning rate to 1.0 would result in strong impact on the preference for the action. Given the stochastic nature of the reward received we do not want to punish an action too hard directly if it has provided us with a lower reward than the baseline reward.

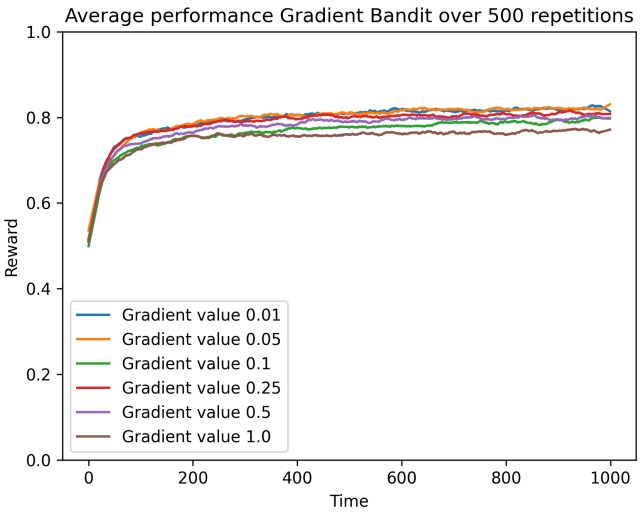

Figure 5: Gradient Bandit algorithm average performance over 500 repetitions for different α values.