Reinforcement Learning in Bandit Tasks and Gridworld

Matiss Kalnare (s3657167) Valeria Rezan (s3504786)
Beyza Celep (s3672468) Catherine Smeyers (s3692566)

October 2024

The complete project is available on our GitHub repository.

1 Introduction

In this report, we address how an agent can learn to make optimal decisions in changing environments using reinforcement learning (RL). Specifically, we look at the multi-armed bandit problem, where an agent must choose between several options ("arms"), each with an unknown probability of reward. The goal of the agent is to maximize the cumulative reward. The challenge lies in finding the balance in exploration - trying various actions to learn about the rewards they lead to - and exploitation - choosing the action that yields the highest reward. This choice of exploration vs exploitation is especially relevant in a changing environment where the reward probabilities change over time.

In real-world situations, either humans or machines are decision-makers, and they often operate in uncertain and ever-changing environments. Think, for example, financial investments, where different assets offer varying returns, or medical decisions, where a certain treatment has uncertain outcomes. The bandit problem gives a simplified yet effective framework for studying how agents can make decisions that maximize long-term rewards while continuously studying the environment. Furthermore, this in a sense mirrors both human and animal learning, particularly learning seen in classical and operant conditioning in psychology. An understanding of this has broad implications, from artificial intelligence to behavioral science.

Our solution here builds upon key developments in machine learning and behavioral science. In particular, we focus on using a weighted average updating rule to adjust an agent's expectations based on the observed rewards and the use of softmax decision rule decision-making for action selection (both will be further discussed in a subsequent section when addressing methodology). Additionally, our work builds upon classical theories of human and animal learning of psychologists like B.F. Skinner, who investigated how rewards and punishments affect behavior. The model uses these principles to find out how different reward

structures (stationary vs volatile) affect an agent's learning and decision-making strategies.

Therefore, we had to not only implement these learning algorithms but also explore how they perform under different conditions. This gives us an insight into how simple rules of both learning and decision-making can lead to adaptive behavior in changing environments.

2 Bandit definition and simulation

In the first part of this assignment, we define and simulate different types of bandits and explore strategies to maximize the rewards we obtain across several trials, given some possible actions. The probability of obtaining a reward from a bandit can be described using a conditional probability distribution $P(r_a|a)$, which maps each action (arm) a to a distribution over possible reward r_a for this action. In this assignment, we simulate four types of bandits with varying reward strategies:

- **Binomial bandit with fixed reward probabilities**
  The binomial bandit returns a reward 1 with probability p.

- **Binomial bandit with walking reward probabilities**
  To simulate the binomial walk, we introduce noise with a mean of 0 and a small standard deviation of 0.05 to the bandit's reward probability. At each timestep t, the reward probability of the bandit is updated as follows: $p_{t+1} = p_t + \mathcal{N}(0, 0.05)$ where p_t is the reward probability at time step t, and $\mathcal{N}(0, 0.05)$ represents Gaussian noise with a mean of 0 and a standard deviation of 0.05. To ensure the reward probabilities remain within the valid range [0, 1], we apply the `np.clip` function, which constrains the updated probabilities to stay within the desired interval.

- **Gaussian bandit with fixed reward probabilities**
  Given a desired mean reward, the Gaussian bandit returns a reward that is calculated using the following formula: $r = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(r-\mu)^2}{2\sigma^2}\right)$, where:

  - μ is the mean reward,
  - σ is the standard deviation (set to 1 in our experiments).

  To implement the rearward calculation we used in-build `np.random.normal` method.

- **Gaussian bandit with walking reward probabilities**
  Similar to the binomial bandit with walking probabilities, the Gaussian bandit with walking reward probabilities is obtained by adding noise with a mean of 0 and a small standard deviation of 0.05 to the current mean value at each time step. This allows the reward probabilities to evolve over time, simulating a dynamic environment. The crucial difference is that the Gaussian walk is not bounded.

Each bandit function accepts a parameter value (the probability for binomial agents and the mean value for Gaussian agents) along with the number of time steps t for which the action will be simulated. The function stores the obtained rewards in an array called `reward_samples` and the corresponding parameters for each time step in a separate `bandit_parameters` array.

For stable bandits, the `bandit_parameters` array will contain the same value across all time steps, reflecting their stable nature. In contrast, for walking bandits, the parameter values will vary at each step due to the introduced noise. To generalize our bandit simulations across multiple bandits, we implemented the `draw_from_k_bandits` function. This function takes the initial parameters and the type of bandit as input and returns both the reward samples and the parameter values for each bandit across all timesteps.

Now, let us return to our main objective: given t possible trials and n agents, we aim to maximize the total reward obtained across these trials. For an agent to select its next action, it must retain information about the average reward received from each action. To facilitate this, we define the action value function $V(a)$, which represents the average reward received from action a:

$$V(a) = E_{r \sim p(r|a)}$$

To make our algorithm more efficient without storing intermediate rewards we can update the value function in two possible ways:

- **Incremental Mean Update:** $V_n = V_{n-1} + \frac{1}{n}[r_n - V_{n-1}]$
- **Learning Update:** $V_n = V_{n-1} + \alpha[r_n - V_{n-1}]$, where the learning rate α is in the interval (0,1). 0.1 in our case.

2.1 The impact of context

To illustrate the difference between incremental and learning updates, we applied these strategies to the four types of 1-arm bandits.

Figure 1 demonstrates that incremental updates of the arm values are well-suited for stable environments. These updates quickly converge to the true value of each bandit and exhibit minimal fluctuations around it. However, in dynamic environments, agents using incremental updates struggle to adapt swiftly. This is because the update factor becomes too small, limiting the agent's ability to respond to changes in the reward distribution.

In contrast, agents using a learning update mechanism perform better in dynamic environments, as their update rate remains constant, allowing them to track changes more effectively. However, this same characteristic leads to poorer performance in stable environments, where the larger updates introduce unnecessary fluctuations around the true value.

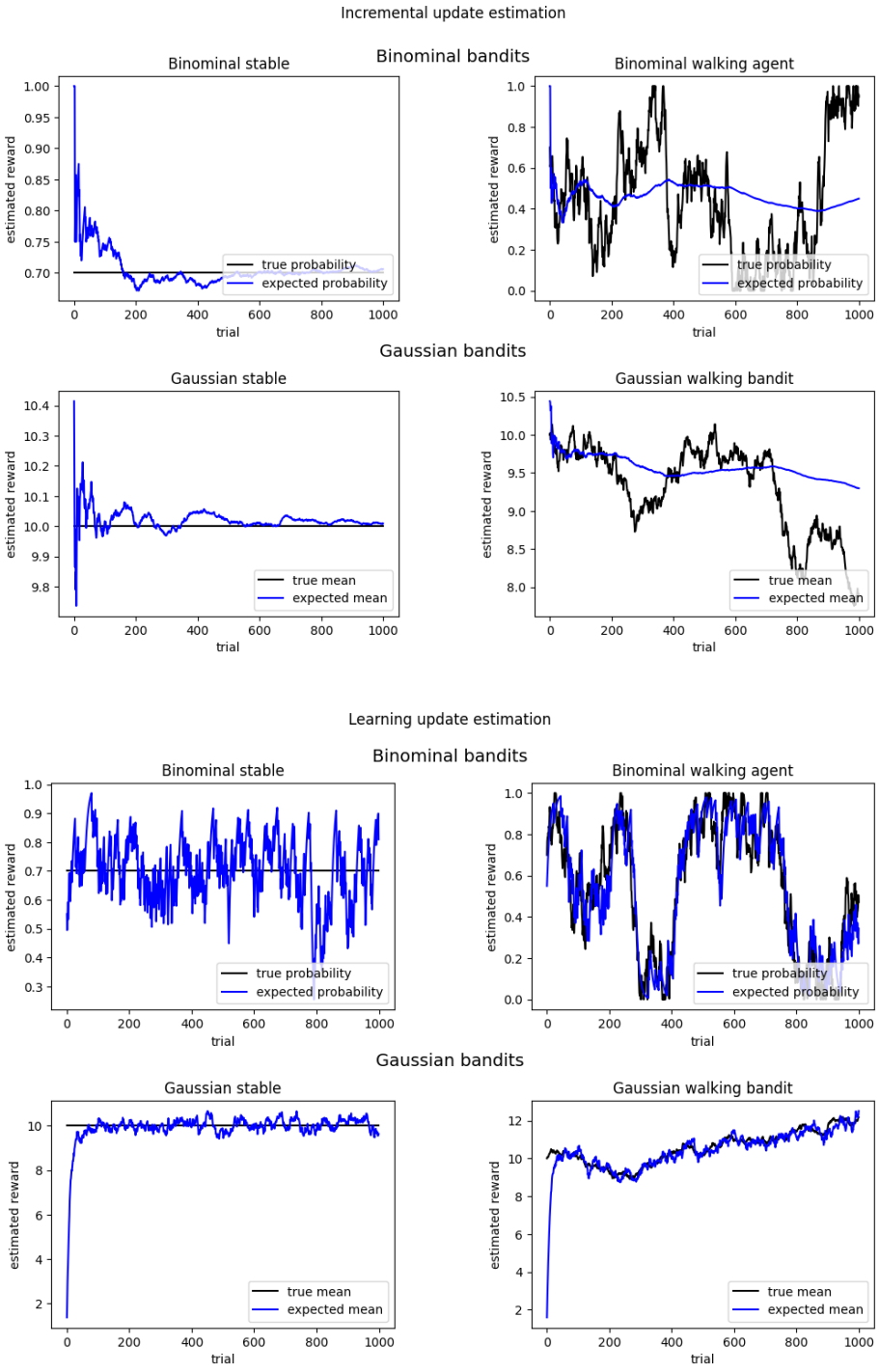

Figure 1: Value estimation function versus the actual value of the bandit in different settings, using the learning and incremental update strategies for Binomial agents with an initial probability value of 0.7 and Gaussian agents with an initial mean value of 10

Let us return to our primary problem. Having learned how to estimate the values of the bandits, the next step is to determine which arm to use at each time step. The most straightforward approach is to select the arm with the highest estimated value. However, this strategy may not lead to optimal results, as it could bias the selection toward the first arm that yields a reward.

To address this issue, we introduce a policy $\pi(a)$ that represents the probability of the agent taking action a and balances exploration and exploitation. In our second experiment, we employed the Softmax policy, which takes the value of the action and a parameter β as arguments and returns a probability for selecting each action using the following formula:

$$\text{Softmax}(a) = \frac{e^{(a)}}{\sum_i e^{\beta V(i)}}$$

The parameter β in this formula balances the exploration and exploitation in the action selection. Specifically, a higher β increases the likelihood of selecting actions with higher estimated values (favoring exploitation), while a lower β promotes exploration by allowing for more equal probabilities across all actions. With all components in place, we can outline the steps of our ultimate recipe for solving the bandit problem:

Algorithm 1 Bandit Problem with Softmax Policy

Simulate rewards for all bandits at time step t
Initialize $V(a)$ for all actions a (e.g., set to zero or random values)
for $t = 1$ to T **do**
    Sample action a_t using Softmax policy and value function V.
    Receive reward r_t from environment for action a_t
    Calculate prediction error: $\delta_t = r_t - V(a_t)$
    Update action-value function: $V(a_t) \leftarrow V(a_t) + \alpha \cdot \delta_t$

At each step, we checked the optimal action based on the expected actual values of the bandits at each time step. We recorded the value in the `optimum_softmax` array as follows:

$$\texttt{optimum\_softmax}_t = \begin{cases} 1, & \text{if the selected action is optimal,} \\ 0, & \text{otherwise.} \end{cases}$$

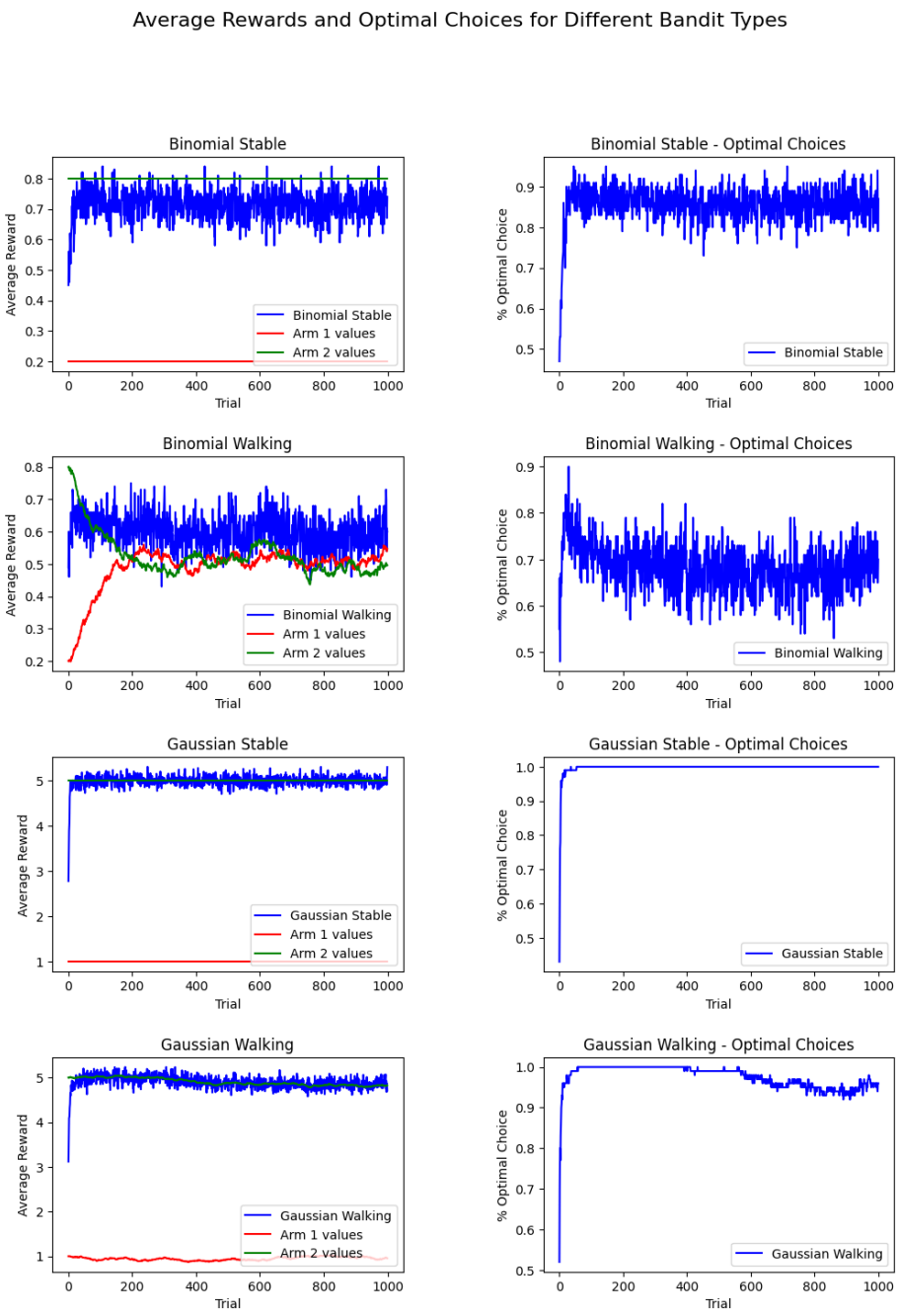

Figure 2: Average reward and percentage of optimal choices for the softmax policy agent using the learning update in various settings of the two-arm bandit problem. The learning rate α and the parameter β are set to 0.1 and 2, respectively. The initial parameters for Binominal models were set to [0.2, 0.8] and for Gaussian models to [1, 5]. The results are averaged over 100 repetitions.

In the stable environment with the two-arm binomial bandit, the agent quickly learns the optimal action after about 50 repetitions, with a probability of 0.8. The mean reward stabilizes around 0.7 since even when the agent selects the optimal arm, it still receives a reward of 0 in 20% of cases. As seen in Figure 3, the agent acts optimally only about 90% of the time because the softmax policy encourages continued exploration of the other action.

In the case of binomial walking bandits, the arm values gradually converge to a 0.5 probability after about 200 iterations. This happens because the walk is bounded, causing the higher-probability arm to decrease while the lower-probability arm increases. Consequently, the mean reward probability drops to around 0.6. Initially, as in the stable environment, the agent learns the optimal action. However, as the action probabilities become closer, it becomes more challenging for the agent to distinguish between them, leading to a decline in optimal choices.

For the stable Gaussian bandit, we initialized the mean values to 1 and 5. As shown in the figure, after around 50 time steps, the agent converges to the optimal arm. The percentage of optimal choices reaches 100% after convergence, suggesting that the agent stops exploring, unlike in the stable binomial environment. The reason for this is that the difference between the value estimates of the arms is significantly larger, leading to a more decisive behavior.

The last row of the figure depicts the agent's behavior in a non-stationary Gaussian bandit environment. The arm values do not deviate much, likely due to the average of the results being over 100 repetitions. Even in this dynamic environment, the agent adapts effectively, quickly learning which decision is optimal. However, as seen in the optimality graph, after 500 iterations, the agent starts to explore sub-optimal values again. This is likely because the mean values of the arms begin to converge, prompting the agent to resume exploration. Note that the actual drift in the arms' mean values is not clearly visible in the plot, due to the generalization across 100 repetitions.

2.2 The interaction of agent and context

In this section, we explore how an agent interacts with a dynamic environment characterized by volatility, which reflects the rate of change in the environment's reward structure. The goal of our experiment is to understand how an agent's learning rate (α), inverse temperature (β), and environmental volatility affect the agent's performance in optimizing rewards over time.

In the softmax function, β acts as an inverse temperature parameter, controlling the action selection. A high β value favors exploitation by concentrating action selection around those with the highest expected rewards. A low β value makes action selection more uniform, leading to more exploration and less reliance on prior rewards. The learning rate (α) controls the speed at which the agent updates its value estimates in response to new rewards. A higher learning rate allows the agent to quickly adjust to changing reward structures. A lower learning rate prevents the agent from overreacting to occasional fluctuations in

rewards (i.e. noise). Volatility in our environment represents the rate of change in reward probabilities. Higher volatility means that the optimal action changes frequently, challenging the agent to continuously reassess its strategy. Therefore, in such environments, agents must rely more on exploration and rapid learning to stay responsive to the shifting reward structure.

With the above considerations in mind, we form a hypothesis:

- In stable environments (low volatility): Agents benefit from exploitation. A higher β and a lower learning rate allow the agent to settle on an optimal strategy and achieve high rewards, as the reward structure does not change significantly over time.
- In volatile environments (high volatility): Exploration becomes essential. Agents with a lower β and a higher learning rate can adapt to the shifting reward landscape more effectively, as they continuously reassess their strategy based on new information.

We conducted a grid-search type experiment with β values of 1, 3, and 5 along with α values of 0.01, 0.1, and 0.5 in three different environments, namely one with volatility of 0.01, one with 0.5, and lastly with 1. The result of our experiment can be seen in Figure 3.

Figure 3: Heatmaps of average reward performance for varying learning rates and β values across different volatility's across 1000 trials for 500 agents. A lighter color indicates better performance.

It is important to note that the resulting graphs do not give clear indications on which parameters are the best for each environment, as the results vary on each run of the experiment. However, a general trend could be noticed. Considering the low volatility environment, we notice that the best-performing agent has the following parameters: $\beta = 3$ and α=0.5 which does not exactly align with our hypotheses. However, it is important to notice the scale on which the average reward operates (the difference between the best and worst performing agent is 0.15). For the best-performing agent that would align with our hypotheses, the average reward is only lacking behind about 0.1 or 0.2 points. In the

moderate volatility environment, we see that the best-performing agent has the lowest β value suggesting exploration is needed, as well as the highest learning rate, indicating that the environment changes fast and adaptations are needed quickly. Similar observation can be made for the high-volatility environment, where a clear stand-out best-performing agent is noticeable.

Furthermore, for both moderate and high-volatility environments, we see that the performance of agents that do not align with our hypotheses perform visibly worse. Thus we can conclude, that agents with higher learning rates and lower β values perform best in environments with high volatility, while agents with lower learning rates and higher β values excel in more stable environments.

NOTE: *To reiterate, the results from the heat map did not always yield these exact results and at times gave results that did not match exactly to our hypotheses, but a general trend was still there.*

2.3 Alternatives to performance

There are various ways to measure the performance of agents in decision-making tasks. In this section, we will focus on one such measure by analyzing the Q-values, which represent the expected reward of each action. Specifically, we will assess how closely these Q-values align with the actual reward probabilities. The objective is for the agent's Q-values to approximate the true reward probabilities as accurately as possible. For a bandit problem this would mean that if the estimated value of one bandit is close to its reward probability, the agent performs well. In the following experiment, we run an agent for a two-armed bandit problem, where the bandit has a changing reward probability over the 1000 trials. In one case, the probability changes according to Gaussian walking; in the second, it is updated using binomial walking.

When looking at the graphs we can see that for both Gaussian walking and binomial walking the agent shows a correlation between the Q-values and the actual reward probabilities. Their curves resemble the ones of the actual reward probabilities, however it is difficult to assess exactly how well the agents perform.

To better compare the performance of specific agents, it is important to have an exact measure. This is where the correlation coefficient comes in. It allows us to get an overall value indicating how well an agent performed in approximating the reward probabilities over all trials. If the correlation coefficient is 1 it means that the agent perfectly represents the reward probabilities with its Q-values, if it is 0 there is no correlation between the Q-values and the actual rewards at all.

In the following experiment, we run 500 agents with a learning rate of 0.1 all going through 3000 trials within a changing environment using Gaussian walking.

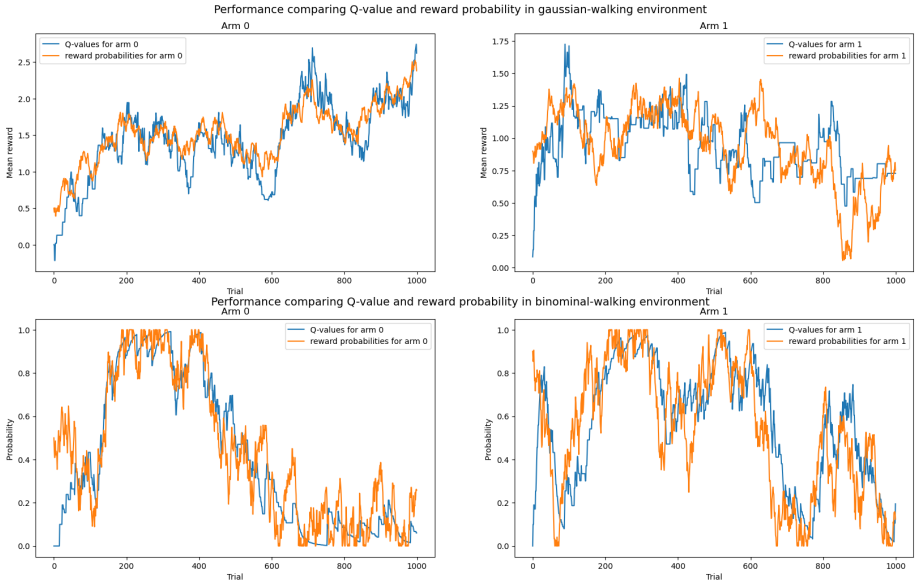

Figure 4: Comparison of agent values and reward probabilities updated using Gaussian walking in each trial over a total of 1000 trials

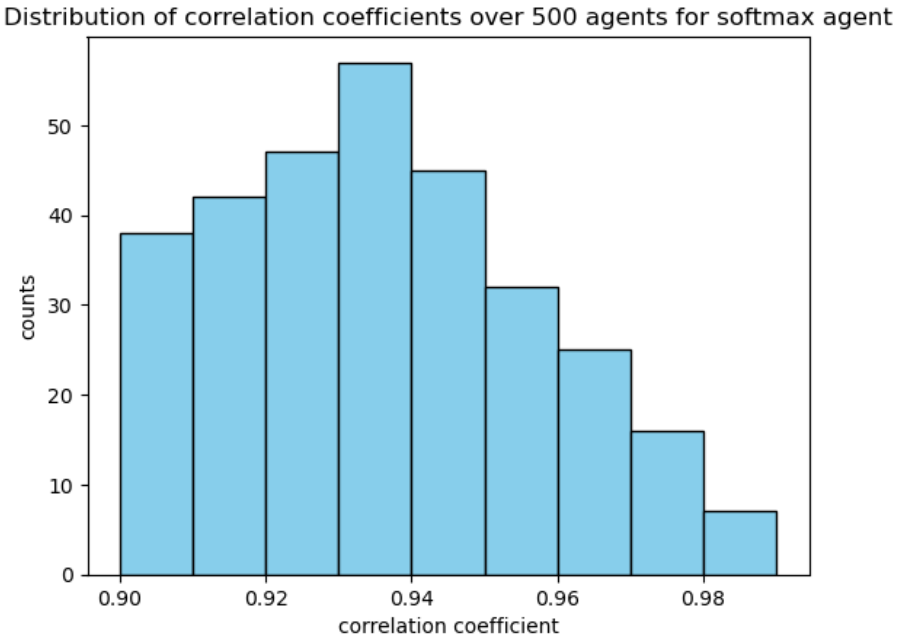

Figure 5: distribution of the total correlation coefficient of 500 agents over 300 trials

The distribution shows an overview of all observed correlation coefficients of the 500 agents. As we can see agents generally perform quite well. The total average correlation coefficient of these agents in an environment with 2 bandit arms is 0.917. This value increases as the number of trials increases as the agent gets more opportunities to learn from the environment.

Next, we want to analyze what can improve the performance of different agents. For this we focus on the Softmax agent, changing its learning rate and β value to optimize the correlation coefficient.

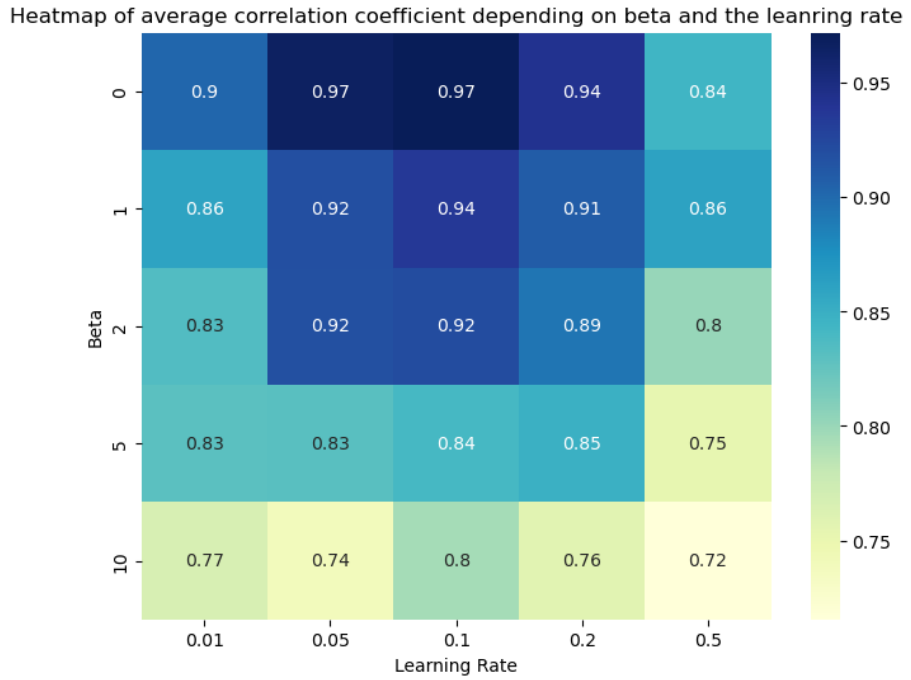

Figure 6: Heatmap of the average correlation coefficient over 10 agents with 3000 trials when varying the learning rate and β value

The heatmap compares the overall performance of different Softmax agents depending on their learning rate and β value. We can observe that the agent performs best with β=0 and learning rate α=0.1, giving us a correlation coefficient of 0.97. This indicates that a pure exploration tactic is as efficient as β=0, which leads to all actions having the same probability. Hence, the agents can update all Q-values equally often. While this might be the best agent to generate Q-vaflues, this does not mean that it necessarily is the agent that will lead to the maximum reward. Even though it describes the actual reward probabilities the best, this agent would not choose the best action leading to the maximal reward.

3 Discussion

In this assignment, we explored the multi-armed bandit problem using reinforcement learning (RL), with the main focus on balancing exploration and exploitation in both static and dynamic environments. To reiterate, we implemented 4 types of bandits – binomial and Gaussian models, both with fixed and walking reward probabilities, to simulate the effects of change in the environment on the agents' performance. In general, we utilized the weighted-average updating rule for value estimation along with the Softmax policy for action selection.

However, we also investigated the performance differences of two update functions – **incremental mean update** and the **learning rate update**. We found that the incremental mean update led to stable performance in environments with fixed reward probabilities but was less effective in dynamic environments due to its slower adaptation. Conversely, the learning rate update allowed for faster adaptation, making it better suited for volatile environments, but less so for stable ones due to the increased fluctuations.

Furthermore, we found that a lower learning rate and higher β are better in stable environments, whereas, in dynamic environments, a higher learning rate helps with adapting, while lower β ensures exploration of the everchanging options.

These above-mentioned findings are significant because they demonstrate that the optimal strategy for reinforcement learning agents depends on the specific characteristics of the environment. In stable settings, certain parameter choices lead to better performance, while in dynamic environments, different choices are more effective. This underscores the need to adapt learning strategies to the environment rather than relying on a *one-size-fits-all* approach.

While our results provide some valuable insights, there is always space for improvement. For example, more extensive hyperparameter tuning. This could be done using various modern approaches such as evolutionary algorithms or Bayesian optimization, which is far more computationally efficient than our simple grid search. Furthermore, it could be beneficial to experiment with the use of advanced RL algorithms like Upper Confidence Bound (UCB) alongside Softmax in different environments for a more robust exploration strategy. Additionally, we did not account for the potential effects of different reward distributions (e.g., Poisson) on performance, which can be explored in future work.

Finally, further research could investigate hybrid models, that mayhaps combine incremental and learning rate updates and/or hybrid action selection strategies, to strike the perfect balance between stability and adaptability in more complex environments. An interesting extension would be to apply these principles to real-world scenarios, where environmental volatility is extremely common, to asses the practical applicability of our findings.