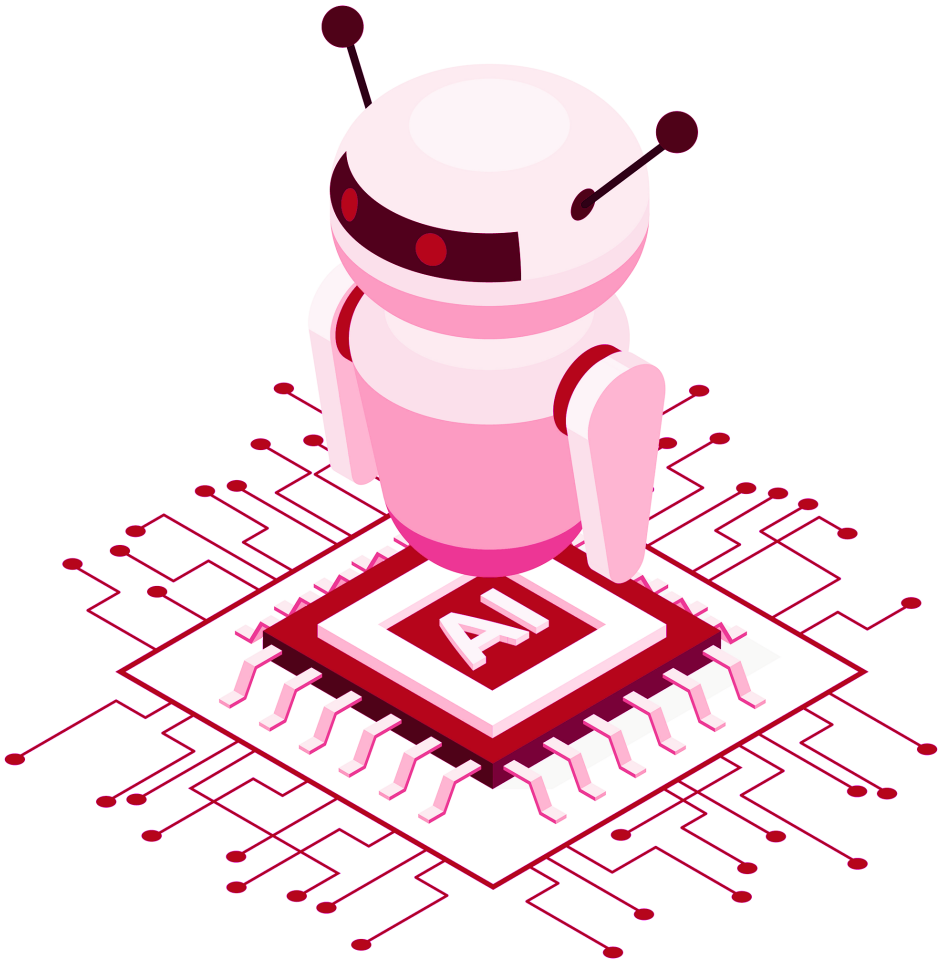

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 6:

Multi-Armed Bandits

By:

Danial Parnian

401110307

Spring 2025

Contents

Grading

The grading will be based on the following criteria, with a total of 105.25 points:

Task	Points
Task 1: Oracle Agent	3.5
Task 2: Random Agent	2
Task 3: Explore-First Agent	5.75
Task 4: UCB Agent	7
Task 5: Epsilon-Greedy Agent	2.25
Task 6: LinUCB Agent	27.5
Task 7: Final Comparison and Analysis	6
Task 8: Final Deep-Dive Questions	41.25
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

0 Task 0: MAB Overview

- Questions:
  - How might the performance of different agents change if the distribution of probabilities were not uniform?

    Agents that rely on uniform exploration (like Explore-First and Epsilon-Greedy) may perform worse, because they would spend a lot of time exploring suboptimal arms.
  - Why does the MAB environment use a simple binary reward mechanism (1 or 0)?

    Probably for simplicity. No extra complexity is added to problem (like continuous distributions for rewards), and we can easily focus on exploration-exploitation tradeoff.

1 Task 1: Oracle Agent

- The Oracle uses privileged information to determine the maximum expected reward.
- **TODO:** Compute the oracle reward (2 points).
- Questions:
  - What insight does the oracle reward provide? (0.75 points)

    It indicates the reward after choosing the best arm, and therefore is an upper bound on performance.
  - Why is the oracle considered "cheating"? (0.75 points)

    Because in real world problems, we don't know the expected reward of arms, and the whole point of MAB is to estimate those and find optimal arm. So we shouldn't rely on oracle.

2 Task 2: Random Agent (RndAg)

- This agent selects actions uniformly at random.
- **TODO:** Choose a random action (1 point).
- Questions:
  - Why is its reward lower and highly variable? (0.25 points)

    Because in real world problems, we don't know the expected reward of arms, and the whole point of MAB is to estimate those and find optimal arm. So we shouldn't rely on oracle.
  - How could the agent be improved without learning? (0.75 points)

    we can implement a weighted random selection, if we have any information about the environment and rewards. For example if we know some arms are better, assign them higher probability.

3 Task 3: Explore-First Agent (ExpFstAg)

- The agent explores randomly for a fixed number of steps and then exploits the best arm.

- **TODOs:**
  - Update Q-value (3 points)
  - Choose action based on exploration versus exploitation (1 point)
- Questions:
  - Why might the early exploration phase (e.g., 5 steps) lead to high fluctuations in the reward curve? (0.75 points)

    Because the agent is selecting arms randomly, to gather information and have better estimates, and is still not using its information. Therefore, it selects all arms (almost) equivalently which leads to fluctuations in the reward curve.

  - What are the trade-offs of using a fixed exploration phase? (1 point)

    It is simple to implement and helps us have prior knowledge on all arms, but on the other hand, the exploration may not be enough to have good estimates, especially if reward distributions are noisy. Also, it can be inefficient, if some arms have terrible rewards and easily estimated, but we would still pick them a constant number in the exploration phase and intentionally delay the exploitation. So we have to carefully tune exploration values and even with that, still might face the mentioned challenges.

  - How does increasing max_ex affect the convergence of the agent's performance?

    It delays the initial convergence, and forces agent to spend more time exploring (which could be either helpful and harmful).

  - In real-world scenarios, what challenges might arise in selecting the optimal exploration duration?

    The optimal exploration duration depends on the reward distribution, which is often unknown. Also, if the reward distributions change over time, we can't use a fixed exploration duration. In general, determining when the exploration phase is enough, and we should enter exploitation, is difficult and setting the value too high or too low is dangerous.

4 Task 4: UCB Agent (UCB_Ag)

- Uses an exploration bonus to balance learning.
- **TODOs:**
  - Update Q-value (3 points)
  - Compute the exploration bonus (4 points)

5 Task 5: UCB vs ExpFstAg

- Questions:
  - Why does UCB learn slowly (even after 500 steps, not reaching maximum reward)? *Hint:* Consider the conservative nature of the exploration bonus.

    Because it balances exploration and exploitation and tries to explore only high-potential arms or less-picked ones, and it may take long to be able to find optimal arm. But this conservative attitude could be good, especially because it avoids randomly picking terrible arms just for the

sake of exploration, and performs a smarter exploration.

- Under what conditions might an explore-first strategy outperform UCB, despite UCB's theoretical optimality?

  In scenarios when the exploration phase of Explore-First algorithm quickly finds the optimal arm (for example when environment is too simple and reward distributions are simple, like here which is Bernoulli), it would converge faster than UCB.

- How do the design choices of each algorithm affect their performance in short-term versus long-term scenarios?

  Explore-First is better in short term, since it quickly estimates value of different arms. But it may perform worse in long term if the exploration phase is insufficient. UCB balances short and long term performances by being conservative, but still exploring high potential arms. But comparing to Explore-First, it starts slower and overtakes over time.

- What happens if we let ExpFstAg explore for 20 steps? Compare its reward to UCB.

  It finds more accurate estimates for q-values and therefore quickly finds optimal arm. As you can see in the plot, it outperforms UCB rapidly.

- What impact does increasing the exploration phase to 20 steps have compared to 5 steps?

  It increases the exploration phase duration, which leads to a more accurate estimate of arm rewards, and rapidly find the optimal arm. As you can see in the plot, it even outperforms UCB in only 20 steps.

- How can you determine the optimal balance between exploration and exploitation in practice?

  How can you determine the optimal balance between exploration and exploitation in practice?

- We know that UCB is optimal. Why might ExpFstAg perform better in practice? (Discuss how hyperparameter tuning and early exploitation can sometimes yield higher rewards in finite-time scenarios.)

  Due to early exploitation. A well tuned exploration phase in ExpFstAg can quickly identify a near-optimal arm, which allows faster exploitation compared to UCBs more cautious exploration, and outperform it in finite-time scenarios.

6 Task 6: Epsilon-Greedy Agent (EpsGdAg)

- Selects the best-known action with probability $1 - \varepsilon$ and a random action with probability ε.
- **TODO:** Choose a random action based on ε (1 point)
- Questions:
  - Why does a high ε result in lower immediate rewards? (0.5 points)

Because the agent explores more often, selecting suboptimal actions with higher probability instead of exploiting the current best estimate, which leads to lower rewards.

- What benefits might decaying ε over time offer? (0.75 points)

  Decaying ε over time allows for initial exploration to find accurate reward estimates, and then increase exploitation as the agent becomes more confident about its estimates. This balances exploration and exploitation effectively.

- How do the reward curves for different ε values reflect the exploration–exploitation balance? (1.25 points)

  ε=0 is the worst because we don't explore at all. ε=0.1 balances exploration and exploitation effectively and achieves the best result, amd increasing ε more than that leads to damage to performance over time (less exploitation).

- Under what circumstances might you choose a higher ε despite lower average reward? (1.25 points)

  When the environment is non-stationary or when there is high uncertainty about the true reward distributions. In such cases we prefer more and continuous exploration, encouraging us to pick higher epsilon.

7 Task 7: LinUCB Agent (Contextual Bandits)

- Leverages contextual features using a linear model.
- **TODOs:**
  - Compute UCB for an arm given context (7 points)
  - Update parameters A and b for an arm (7 points)
  - Compute UCB estimates for all arms (7 points)
  - Choose the arm with the highest UCB (4 points)
- Questions:
  - How does LinUCB leverage context to outperform classical bandit algorithms? (1.25 points)

    it incorporates state features into the rewards estimates, and considers the context, unlike classical algorithms that treat each arm independently.

  - What is the role of the α parameter in LinUCB, and how does it affect the exploration bonus? (1.25 points)

    It controls the exploration term. A higher α leads to more exploration by increasing effect of UCBs. A lower α leads to more exploitation by reducing that term.

  - What does α affect in LinUCB? (1.25 points)

    Higher alpha encourages more exploration and therefore finding more accurate estimates. But increasing it more than a certain amount, might delay exploitation and lead to worse short-term performance.

  - Do the reward curves change with α? Explain why or why not. (1.25 points)

    α=0.01 is the best value and balances between exploration and exploitation well. higher values delay the exploitation and decrease rewards in short term, and α=0 has little exploration which leads to weak performance. So the reward curves change, as you can see.

- Based on your experiments, does LinUCB outperform the standard UCB algorithm? Why or why not? (1.25 points)

  It's difficult to pick one of them! it totally depends on the problem. LinUCB uses contextual information, and if we have relevant contexts, and also the linear combination of them could determine reward, LinUCB could use these data and perform better. When context isn't informative, we could use simple UCB.

- What are the key limitations of each algorithm, and how would you choose between them for a given application? (1.25 points)

  LinUCB relies on a linear relationship between context and reward, which may not hold. It can also suffer from overfitting in high-dimensional contexts. UCB Ignores context, which can lead to suboptimal decisions when context is informative. when context is highly informative and a linear relationship is plausible, it's better to use LinUCB. otherwise, UCB is the best option.

8 Task 8: Final Deep-Dive Questions

- **Finite-Horizon Regret and Asymptotic Guarantees** (4 points)

  Many algorithms (e.g., UCB) are analyzed using asymptotic (long-term) regret bounds. In a finite-horizon scenario (say, 500–1000 steps), explain intuitively why an algorithm that is asymptotically optimal may still yield poor performance. What trade-offs arise between aggressive early exploration and cautious long-term learning? Deep Dive: Discuss how the exploration bonus, tuned for asymptotic behavior, might delay exploitation in finite time, leading to high early regret despite eventual convergence.

  In a finite-horizon scenario an optimal algorithm like UCB may perform poorly because its exploration bonus, tuned for the long term, can delay exploitation, leading to high early regret despite eventual convergence. Aggressive early exploration may quickly identify the best arm and cautious long-term learning balances exploration and exploitation over a longer time.

- **Hyperparameter Sensitivity and Exploration–Exploitation Balance** (4.5 points)

  Consider the impact of hyperparameters such as ϵ in ϵ-greedy, the exploration constant in UCB, and the α parameter in LinUCB. Explain intuitively how slight mismatches in these parameters can lead to either under-exploration (missing the best arm) or over-exploration (wasting pulls on suboptimal arms). How would you design a self-adaptive mechanism to balance this trade-off in practice? Deep Dive: Provide insight into the "fragility" of these parameters in finite runs and how a meta-algorithm might monitor performance indicators (e.g., variance in rewards) to adjust its exploration dynamically.

  If hyperparameters are too small, we may underexplore and dont find accurate estimates, and therefore choosing and unoptimal arm. In the other hand, setting them too high leads to overexplore and despite finding accurate enough estimates, we may force the model to explore more and not exploit, which leads to higher unnecessary regret. A self-adaptive mechanism could monitor performance indicators like reward variance to dynamically adjust exploration. For example, high reward variance indicates the need for more exploration, while stable rewards indicate it's time to exploit.

- **Context Incorporation and Overfitting in LinUCB** (4 points)

LinUCB uses context features to estimate arm rewards, assuming a linear relation. Intuitively, why might this linear assumption hurt performance when the true relationship is complex or when the context is high-dimensional and noisy? Under what conditions can adding context lead to worse performance than classical (context-free) UCB? Deep Dive: Discuss the risk of overfitting to noisy or irrelevant features, the curse of dimensionality, and possible mitigation strategies (e.g., dimensionality reduction or regularization).

When the true relationship between context and reward is complex and non-linear, LinUCB may fail to estimate q-values, which leads to bad performance. In high-dimensional and noisy contexts, the linear assumption can lead to overfitting, where the model fits the noise rather than the underlying relationship. Adding context may lead to worse performance than classical UCB when the context is irrelevant or noisy or when the linear assumption is a poor fit for the true relationship. One solution is to use dimensionality reduction techniques (like PCA) to reduce number of features and noise. We can also use regularization to prevent overfitting.

- **Adaptive Strategy Selection** (4.25 points)

  Imagine designing a hybrid bandit agent that can switch between an explore-first strategy and UCB based on observed performance. What signals (e.g., variance of reward estimates, stabilization of Q-values, or sudden drops in reward) might indicate that a switch is warranted? Provide an intuitive justification for how and why such a meta-strategy might outperform either strategy alone in a finite-time setting. Deep Dive: Explain the challenges in detecting when exploration is "enough" and how early exploitation might capture transient improvements even if the long-term guarantee favors UCB.

  A hybrid bandit agent can switch between explore-first and UCB based on performance signals. High variance in reward estimates, unstable Q-values, or sudden reward drops suggest more exploration is needed. Stabilization of Q-values or consistently high rewards indicates it's time to exploit. This strategy can outperform both algoritms by adapting to different phases of the learning process and take the good part of both algorithms (explore first to quickly find good estimates, and then UCB to balance exploitation and exploration for the rest).

- **Non-Stationarity and Forgetting Mechanisms** (4 points)

  In non-stationary environments where reward probabilities drift or change abruptly, standard bandit algorithms struggle because they assume stationarity. Intuitively, explain how and why a "forgetting" or discounting mechanism might improve performance. What challenges arise in choosing the right decay rate, and how might it interact with the exploration bonus? Deep Dive: Describe the delicate balance between retaining useful historical information and quickly adapting to new trends, and the potential for "chasing noise" if the decay is too aggressive.

  A forgetting mechanism improves performance by down-weighting past rewards, which allows the algorithm to adapt to changing reward probabilities. Choosing the right decay rate is crucial: a high decay rate may lead to chasing noise, while a low rate might delay adaptation. The decay rate determines how quickly the algorithm trusts recent rewards over older ones, and balances between exploration and exploitation balance.

- **Exploration Bonus Calibration in UCB** (3.75 points)

  The UCB algorithm adds a bonus term that decreases with the number of times an arm is pulled. Intuitively, why might a "conservative" (i.e., high) bonus slow down learning—even if it guarantees

asymptotic optimality? Under what circumstances might a less conservative bonus be beneficial, and what risks does it carry? Deep Dive: Analyze how a high bonus may force the algorithm to continue sampling even when an arm's estimated reward is clearly suboptimal, thereby delaying convergence. Conversely, discuss the risk of prematurely discarding an arm if the bonus is too low.

A conservative (high) bonus in UCB can slow down learning by forcing continued sampling of suboptimal arms, delaying convergence, even with asymptotic optimality guarantees. A less conservative bonus can be beneficial when quick convergence is prioritized, but it has the risk of prematurely discarding potentially good arms due to under exploration.

- **Exploration Phase Duration in Explore-First Strategies** (4 points)

  In the Explore-First agent (ExpFstAg), how does the choice of a fixed exploration period (e.g., 5 vs. 20 steps) affect the regret and performance variability? Provide a scenario in which a short exploration phase might yield unexpectedly high regret, and another scenario where a longer phase might delay exploitation unnecessarily. Deep Dive: Discuss how the "optimal" exploration duration can depend heavily on the underlying reward distribution's variance and the gap between the best and other arms, and why a one-size-fits-all approach may not work in practice.

  A short exploration can lead to high regret if the initial samples guide the agent towards false reward distributions (happens when distributions are more complex and wide). A longer exploration phase might delay exploitation unnecessarily if the optimal arm is quickly identified (like in a simple environment). The optimal exploration duration depends on the reward distribution's variance and the gap between the best and other arms. As explained above, if the optimal arm is easily found, we shouldn't explore too much and delay the exploitation, also in high variance scenarios we need more exploration. Therefor a one-size-fits-all approach isn't good and we may act based on the problem.

- **Bayesian vs. Frequentist Approaches in MAB** (4 points)

  Compare the intuition behind Bayesian approaches (such as Thompson Sampling) to frequentist methods (like UCB) in handling uncertainty. Under what conditions might the Bayesian approach yield superior practical performance, and how do the underlying assumptions about prior knowledge influence the exploration–exploitation balance? Deep Dive: Explore the benefits of incorporating prior beliefs and the risk of bias if the prior is mis-specified, as well as how Bayesian updating naturally adjusts the exploration bonus as more data is collected.

  Bayesian methods (like Thompson Sampling) use prior beliefs to guide exploration, while frequentist methods (like UCB) rely on observed data. Bayesian approaches can be better when you have good prior knowledge. The accuracy of prior knowledge affects how well Bayesian methods balance exploration and exploitation.

- **Impact of Skewed Reward Distributions** (3.75 points)

  In environments where one arm is significantly better (skewed probabilities), explain intuitively why agents like UCB or ExpFstAg might still struggle to consistently identify and exploit that arm. What role does variance play in these algorithms, and how might the skew exacerbate errors in reward estimation? Deep Dive: Discuss how the variability of rare but high rewards can mislead the agent's estimates and cause prolonged exploration of suboptimal arms.

  UCB (or ExpFstAg) struggle because high variance in rewards can mislead the agent, causing it to explore suboptimal arms longer. UCB tends to explore less-picked arms more to make estimates

more accurate, and Explore-First just randomly picks them more than needed (because we can quickly notice these arms aren't useful due to small q-values). Skewed probabilities and rare high rewards can distort reward estimation, delaying the identification of the best arm.

- **Designing for High-Dimensional, Sparse Contexts** (5 points)

  In contextual bandits where the context is high-dimensional but only a few features are informative, what are the intuitive challenges that arise in using a linear model like LinUCB? How might techniques such as feature selection, regularization, or non-linear function approximation help, and what are the trade-offs involved? Deep Dive: Provide insights into the risks of overfitting versus underfitting, the increased variance in estimates from high-dimensional spaces, and the potential computational costs versus performance gains when moving from a simple linear model to a more complex one.

  In contextual bandits with many features but only a few being useful, LinUCB faces challenges because it tries to learn weights for all features, which can lead to overfitting and poor performance (This is also a challenge in classic classification ML problems with high dimensional feature vectors). To address the problem, feature selection helps by focusing on the most relevant features, regularization prevents overfitting by penalizing large weights, and Non-linear models can capture complex relationships (but are computationally expensive). For example we can use a neural network, which increases the complexity of problem. The trade-offs involve balancing model complexity, computational cost, and the risk of overfitting versus underfitting.