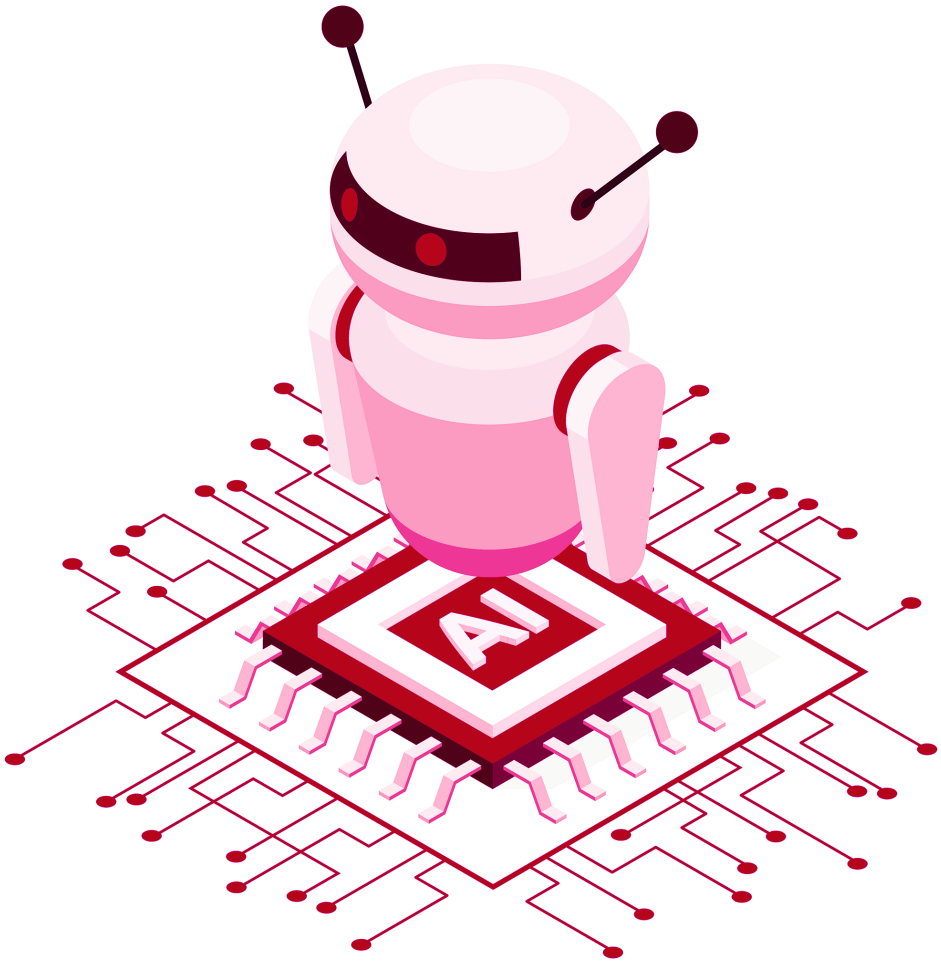

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 6:

Multi-Armed Bandits

By:

Amir Kooshan Fattah Hesari

401102191

RIML

Deep RL Course

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

1 Task 1: Oracle Agent

- The Oracle uses privileged information to determine the maximum expected reward.
- **TODO:** Compute the oracle reward (2 points).
  It was done in the notebook.
- Questions:
  - What insight does the oracle reward provide? (0.75 points)
    Comparing the reward obtained by the bandit algorithm to the oracle reward can help us assess how good is our algorithm and try to improve it.
  - Why is the oracle considered "cheating"? (0.75 points)
    The oracle is considered "cheating" in the context of the Multi-Armed Bandit (MAB) problem because it provides perfect knowledge of the reward distribution for each arm, something that is not available to the agent during the actual decision-making process.

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 the agent isn't learning anything and is choosing a random action in each step.
  - How could the agent be improved without learning? (0.75 points)
    If some prior knowledge about the arms is available, even without full learning, the agent can exploit that and improve the gained reward.

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 a short exploration phase lead to fluctuations? (0.75 points)
    A short exploration phase might lead to fluctuations because the agent explores randomly for only a limited number of steps, which may not be sufficient to fully identify the best arm. During this brief exploration, the agent could encounter bad rewards from an arm that actually has the highest expected return. However, because the exploration phase is short, the agent might not gather enough information about the true value of this arm. As a result, it may under-exploit this arm during the subsequent exploitation phase, leading to suboptimal rewards and fluctuations.
  - What are the trade-offs of using a fixed exploration phase? (1 point)
    A fixed exploration phase is more efficient in terms of computational resources because it doesn't require continual exploration throughout the entire process. It allows the agent to quickly focus on exploiting the arms once the exploration phase is over. **But** it may not be long enough to properly identify the best arm, especially if there is a large variance in rewards or if some arms require more exploration to estimate their true value.

  - (Question in the notebook) How does increasing max-ex affect the convergence of the agent's performance?
    It might take some more time , but the results are better and converge to the real value with higher max-ex.
  - (Question in the notebook) In real-world scenarios, what challenges might arise in selecting the optimal exploration duration?
    The reward distributions of the arms might change over time due to external factors or evolving system conditions, making it hard to predict the optimal exploration duration. What worked initially may no longer be optimal later.

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)
- Questions:
  - Under what conditions might an explore-first strategy outperform UCB, despite UCB's theoretical optimality?
    When we have the following problems, an explore-first strategy outperforms UCB :
    * Highly Noisy Environments with high variance.
    * High Cost of Exploration, making UCB's balance of exploration and exploitation inefficient.
    * Small Number of Arms, where exploration is quick and the optimal arm can be identified early.
    * Limited Time or Resources for exploration, which necessitates a more structured approach.

    The key idea is that UCB's continual exploration might not be necessary or optimal when exploration can be done upfront in a more controlled and efficient way.
  - How do the design choices of each algorithm affect their performance in short-term versus long-term scenarios?
    Epsilon-Greedy:
    * *Short-term:* May perform poorly due to random exploration; potentially high variability in rewards.
    * *Long-term:* Converges well as it focuses more on exploitation over time, but might not fully exploit the best arm if epsilon is too high.

    UCB (Upper Confidence Bound):
    * *Short-term:* Initially takes longer to identify the best arm due to exploration based on uncertainty.
    * *Long-term:* Performs well in the long run by balancing exploration and exploitation, leading to near-optimal performance.

    Thompson Sampling:
    * *Short-term:* Adapts quickly to the best arm, showing lower variability and faster convergence.
    * *Long-term:* Continues to balance exploration and exploitation efficiently, yielding optimal performance over time.

    Explore-First Strategy:
    * *Short-term:* Performs well if the exploration phase is sufficient to identify the best arm quickly.

* *Long-term:* May suffer if exploration was not thorough enough, potentially exploiting suboptimal arms.

Greedy Algorithm:

* *Short-term:* Can be very effective if the best arm is identified early.
* *Long-term:* Risk of suboptimal performance if the initial selection was wrong, due to no exploration.

Random Agent:

* *Short-term:* Likely to perform poorly with high variability since it doesn't strategically explore.
* *Long-term:* Continues to perform poorly as it doesn't optimize its choices based on past rewards.

– Why does UCB learn slowly (even after 500 steps, not reaching maximum reward)?
because it balances exploration and exploitation, giving more weight to arms with higher uncertainty

5 Task 5: 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 it chooses action more randomly, exploring more and not exploiting the best action until now.
  – What benefits might decaying ε over time offer? (0.75 points)
  It can help us reduce the exploration and go toward the best action at the end and maximize our return.

6 Task 6: 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 methods? (1.25 points)
  LinUCB improves upon classical methods like UCB by incorporating **contextual information** into its decision-making process. While classical UCB relies only on historical rewards and the uncertainty of each arm, LinUCB uses **contextual features** (such as user attributes, time of day, etc.) to model the reward distribution for each arm.
  Reward Model: In LinUCB, the reward for arm a in context x is modeled linearly as:

$$r_a(x) = \theta^T x_a + \epsilon_a$$

where:

* x_a is the context (feature vector) of arm a,

* θ is a parameter vector representing the relationship between the context and the expected reward,
* ϵ_a is noise (random error).

LinUCB computes an upper confidence bound that incorporates the context, helping it balance exploration and exploitation more effectively:

$$\hat{r}_a(x) + \alpha\sqrt{x_a^T(A_a^{-1})x_a}$$

where:

* $\hat{r}_a(x)$ is the predicted reward for arm a in context x,
* A_a is a matrix that captures the covariance of the arm's reward estimates,
* α is a parameter that controls the exploration-exploitation trade-off.

By incorporating context into the upper confidence bound, LinUCB can more effectively explore arms that are likely to perform better under specific conditions, leading to better decision-making compared to classical methods.

– What role does the α parameter play in the exploration bonus? (1.25 points)
It helps with the exploration.

7 Task 7: Final Comparison and Analysis

- **Comparison:** UCB vs. Explore-First agents.
  – Under what conditions might an explore-first strategy outperform UCB? (1.25 points)
  Explore-first might outperform UCB in environments with highly noisy or uncertain reward distributions, where early exploration helps gather more reliable information about the arms quickly.
  – How do design choices affect short-term vs. long-term performance? (1.25 points)
  Design choices, such as exploration duration, can enhance short-term exploration at the cost of long-term exploitation, or vice versa. Too much early exploration can delay convergence to the optimal arm, while too little may miss valuable arms.
- Impact of extending the exploration phase (e.g., 20 vs. 5 steps). (1.5 points total)
  A longer exploration phase (20 steps) may reduce early regret but risk delaying exploitation, while a shorter phase (5 steps) might miss crucial information about the best arm but lead to faster exploitation.
- Discussion on why ExpFstAg might sometimes outperform UCB in practice. (2 points)
  ExpFstAg might outperform UCB in environments with high variability in rewards, as its fixed exploration phase allows for faster identification of the optimal arm before settling into exploitation, avoiding suboptimal arms.

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.

Asymptotically optimal algorithms may perform poorly in finite time due to aggressive early exploration causing high regret. Exploration bonuses tuned for long-term performance can delay exploitation and lead to high early regret, despite eventual convergence.

- **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.

  Mismatched hyperparameters, such as a too-high exploration constant, may cause under-exploration, while too-low values might lead to over-exploration. A self-adaptive mechanism could dynamically adjust exploration based on observed performance metrics like reward variance.

- **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).

  A linear assumption in LinUCB may hurt performance if the true reward structure is non-linear or if the context is noisy or high-dimensional. Adding context can worsen performance due to overfitting, especially if irrelevant features are included.
- **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 agent could switch between strategies based on variance or reward stabilization. The challenge is detecting when exploration is sufficient and balancing early exploitation to capture transient improvements without sacrificing long-term optimality.
- **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 can help in non-stationary environments by discounting outdated information. However, choosing the right decay rate is tricky—too aggressive a decay might lead to chasing noise, while too slow a decay might hinder adaptation.
- **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 high exploration bonus can slow down learning by keeping exploration active even for suboptimal arms, delaying convergence. A lower bonus may accelerate exploitation but risks prematurely discarding potentially optimal arms.
- **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 phase (5 steps) may miss valuable arms, leading to higher regret, while a long exploration phase (20 steps) might delay exploitation unnecessarily. The optimal duration depends on reward distribution variance and arm gaps.

- **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 naturally adjust to uncertainty through prior beliefs, which can improve performance in practice. However, they risk bias if the prior is mis-specified, whereas UCB focuses purely on empirical data.

- **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.

  In skewed environments, UCB or ExpFstAg might struggle due to the large variance in rare high rewards. The skew exacerbates errors in reward estimates, leading to prolonged exploration of suboptimal arms before identifying the optimal one.

- **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.

  LinUCB faces challenges in high-dimensional, sparse contexts due to overfitting, high variance, and the curse of dimensionality. Feature selection, regularization, and non-linear models can help, but they come with trade-offs in computational cost and performance.