INF8250AE(Fall 2024) : Reinforcement Learning — Given on: Sep 09

Assignment #1 — Due on : Sept 25, 10 pm

- The primary objective of this assignment is to evaluate your comprehension and application of multi-armed bandit algorithms. Through this task, you will demonstrate your ability to understand the core concepts, implement the algorithms, and analyze their performance.
- This is an individual assignment. You are not allowed to discuss the problems with other students.
- Make sure to write your code only in the `.py` files provided. Avoid creating new files. Do not rename the files or functions, as it will interfere with the autograder's ability to evaluate your work correctly. Also, do not change the input or output structure of the functions.
- When Submitting to GradeScope, be sure to submit:
  1. A `'.zip'` file containing all your Python codes (in `.py` format) to the `'Assignment 1 - Code'` section on Gradescope.
  2. A `'pdf'` file that contains your answers to the questions and generated plots to the `'Assignment 1 - Report'` entry.
- Part of this assignment will be auto-graded by Gradescope. You can use it as immediate feedback to improve your answers. You can resubmit as many times as you want. We provide some tests that you can use to check that your code will be executed properly on Gradescope. These are not meant to check the correctness of your answers. We encourage you to write your own tests for this.
- You cannot use ChatGPT or any other code assistance tool for the programming part; however, if you use ChatGPT to edit the grammar in your report, you have to explicitly state it in the report.
- If you have questions regarding the assignment, you can ask for clarifications in Piazza. You should use the corresponding tag for this assignment.

1 Aragorn's Trials: The Quest to Find Middle-earth's True Champion

In the twilight of the Third Age, as shadows deepen over Middle-earth, a dire threat looms over all free peoples. Sauron, the Dark Lord of Mordor, has regained much of his former strength, and his malice spreads across the lands like a rising tide. In these dark times, hope rests on a knife's edge, and the fate of the world depends on the courage and strength of its heroes.

Among these heroes stands Aragorn, son of Arathorn, the rightful heir to the throne of Gondor. As a leader of the Free Peoples, Aragorn understands the gravity of the coming war.

He knows that to stand against Sauron, Middle-earth must place its trust in the mightiest of champions, a hero capable of overcoming the most perilous quests and leading the forces of good to victory.

But the paths of Middle-earth are fraught with danger, and the true strength of each hero is veiled in mystery. From the noble halls of Rivendell to the rugged wilds of Rohan, heroes of all races have answered the call—Men, Elves, Dwarves, and even Hobbits. Yet who among them is worthy to be named the Champion of Middle-earth? Aragorn must decide, for the time of reckoning draws near.

Aragorn's goal is to identify the hero with the highest probability of successfully completing the most dangerous and crucial quests. However, the true potential of each hero is unknown at the outset. Through trials and quests, Aragorn must gather knowledge of their abilities, learning from their successes and failures, to ultimately choose the one hero who will lead the charge against Sauron's dark forces.

As a trusted advisor to Aragorn, you have been entrusted with the sacred duty of devising a strategy to discover the greatest hero of Middle-earth. Through a series of trials, you will guide Aragorn in deciding which hero to send on quests, using both exploration to uncover hidden strengths and exploitation to maximize the power of known talents.

Details:

- Heroes and Quests:
  - A fellowship of N heroes, each a legend in their own right, has assembled to face the trials. Among them are the noble Men of Gondor, the wise Elves of Lothlórien, the stout Dwarves of the Lonely Mountain, and the brave Hobbits of the Shire.
  - The true prowess of each hero is represented by a success probability p_i, reflecting their chances of triumph in any given quest. This probability is not known at the beginning and must be inferred from their performance in the trials.
  - When a hero is chosen to undertake a quest—whether it be navigating the treacherous paths of Mirkwood, retrieving a lost relic from the depths of Moria, or defending the borders of Rohan from marauding Orcs—the outcome is determined by their true success probability p_i, drawn from a *Bernoulli distribution*. A success means the hero has overcome the challenge, while a failure indicates they were not up to the task.
- Quests and Outcomes:
  - In each round, Aragorn must choose one hero to send on a quest. After each quest, you will observe whether the hero succeeded or failed, using this knowledge to guide future decisions.
  - The trials require a delicate balance between exploring the potential of lesser-known heroes and exploiting the abilities of those who have already proven their worth. This balance is crucial, as it will allow Aragorn to gather enough information to make an informed decision about who can lead Middle-earth in the final battle.

– By the end of the trials, Aragorn must identify the hero with the highest estimated probability of success—the one who will become the Champion of Middle-earth, standing by his side as they march towards the ultimate confrontation with Sauron.

We will use the multi-armed bandit (MAB) algorithms to help Aragorn. Please use the `code_middle_earth.zip` project and implement or answer the questions below. Make sure to upload all the `.py` files when you are done.

Heroes

1. (5 points): Completing the `Heroes` class in `heroes.py`

   The `Heroes` class has been partially implemented. Your task is to complete the `attempt_quest` method to fulfill the following requirements:

   - Update the total quests and successes for the given hero.
   - Return the reward of the quest: return `1` for a success and `0` otherwise.

Example Setup: From now on, let's work with a specific set of heroes:

```
heroes = Heroes(total_quests=3000, true_probability_list=[0.35, 0.6, 0.1])
```

In this setup:

- `total_quests=3000` specifies the total number of quests you want to simulate.
- `true_probability_list=[0.35, 0.6, 0.1]` defines the true success probabilities for each hero. The first hero has a 35% chance of success, the second hero has a 60% chance, and the third hero has a 10% chance.

ϵ-greedy Method

2. (20 points): Completing the `eps_greedy` method in `eps_greedy.py`
   (a) (10 points) The `eps_greedy` method has been partially implemented. Your task is to complete the missing parts according to the following requirements:
   - Accurately define the values of `optimal_reward` and `optimal_hero_index` based on the true success probabilities.
   - Implement the ϵ-greedy action-selection strategy, ensuring that the exploration-exploitation balance is properly managed.

- Return the following lists:
  - `rew_record`: A list of rewards received at each attempt.
  - `avg_ret_record`: The running average of rewards at each attempt.
  - `tot_reg_record`: The cumulative regret at each attempt, reflecting the difference between the optimal reward and the actual reward.
  - `opt_action_record`: The percentage of times the optimal hero was selected at each attempt.

(b) Execute the command `python eps_greedy.py`. Be aware that this process may take some time to complete.

(c) (5 points) The first experiment investigates the impact of different ϵ values from the set `[0.2, 0.1, 0.01, 0.]`. Once the experiment is complete, locate and report the file `results/epsilon_greedy_various_epsilons.pdf`. Provide an analysis of the results, focusing on how varying ϵ values influence exploration behavior and overall performance. (Provide your plots and explanation in the `.pdf` report file.)

(d) (5 points) The second experiment explores the effect of optimistic initial values with $\epsilon = 0$. After running the experiment, find and report the file `results/epsilon_greedy_various_init_values.pdf`. Discuss how different initial value estimates impact both exploration and the effectiveness of the policy. (Provide your plots and explanation in the `.pdf` report file.)

Upper-Confidence-Bound (UCB) Method

3. (15 points): Completing the `ucb` method in `ucb.py`

(a) (10 points) The `ucb` method has been partially implemented. Your task is to complete the missing parts according to the following requirements:

- Accurately define the values of `optimal_reward` and `optimal_hero_index` based on the true success probabilities.
- Implement the Upper-Confidence-Bound (UCB) action-selection strategy.
- Return the following lists:
  - `rew_record`: A list of rewards received at each attempt.
  - `avg_ret_record`: The running average of rewards at each attempt.
  - `tot_reg_record`: The cumulative regret at each attempt, reflecting the difference between the optimal reward and the actual reward.
  - `opt_action_record`: The percentage of times the optimal hero was selected at each attempt.

(b) Execute the command `python ucb.py`. Be aware that this process may take some time to complete.

(c) (5 points) Our experiment investigates the impact of different c (coefficient) values from the set `[0.0, 0.5, 2]`. Once the experiment is complete, locate and report

the file `results/ucb_various_c_values.pdf`. Provide an analysis of the results, focusing on how varying c values influence exploration behavior and overall performance. (Provide your plots and explanation in the `.pdf` report file.)

Boltzmann Method

4. (15 points): Completing the methods in `boltzmann.py`
   (a) (10 points) The `boltzmann.py` file has been partially implemented. Your task is to complete the missing parts according to the following requirements:
      - Complete the implementation for the `boltzmann_policy` method that gets an array and temperature value (τ) and returns an index sampled from the Boltzmann policy.
      - Fill the missing parts for the `boltzmann` method:
        - Accurately define the values of `optimal_reward` and `optimal_hero_index` based on the true success probabilities.
        - Implement the Boltzmann action-selection strategy.
        - Return the following lists:
          * `rew_record`: A list of rewards received at each attempt.
          * `avg_ret_record`: The running average of rewards at each attempt.
          * `tot_reg_record`: The cumulative regret at each attempt, reflecting the difference between the optimal reward and the actual reward.
          * `opt_action_record`: The percentage of times the optimal hero was selected at each attempt.
   (b) Execute the command `python boltzmann.py`. Be aware that this process may take some time to complete.
   (c) (5 points) Our experiment investigates the impact of different τ values from the set `[0.01, 0.1, 1, 10]`. Once the experiment is complete, locate and report the file `results/boltzmann_various_tau_values.pdf`. Provide an analysis of the results, focusing on how varying τ values influence the results. (Provide your plots and explanation in the `.pdf` report file.)

Gradient Bandits Method

5. (20 points): Completing the `gradient_bandit` method in `gradient_bandit.py`
   (a) (10 points) The `gradient_bandit` method has been partially implemented. Your task is to complete the missing parts according to the following requirements:
      - Accurately define the values of `optimal_reward` and `optimal_hero_index` based on the true success probabilities.

- Implement the Gradient Bandit action-selection strategy, following the lecture notes. Make sure to update the logits accurately both when the average return is used as the baseline and when it is not.
- Return the following lists:
  - `rew_record`: A list of rewards received at each attempt.
  - `avg_ret_record`: The running average of rewards at each attempt.
  - `tot_reg_record`: The cumulative regret at each attempt, reflecting the difference between the optimal reward and the actual reward.
  - `opt_action_record`: The percentage of times the optimal hero was selected at each attempt.

(b) Execute the command `python gradient_bandit.py`. Be aware that this process may take some time to complete.

(c) (5 points) The first experiment examines the influence of varying α values from the set `[0.05, 0.1, 2]` when the average return is used as a baseline. Upon completion, locate and analyze the results in the file `results/gradient_bandit_various_alpha_values_with_baseline.pdf`, focusing on the effect of different α values on overall performance. (Provide your plots and explanation in the `.pdf` report file.)

(d) (5 points) The second experiment investigates the impact of omitting the baseline while using the same set of α parameters. After completing the experiment, find and review the results in the file `results/gradient_bandit_various_alpha_values_without_baseline.pdf`, and discuss how the inclusion or exclusion of the baseline affects the policy's effectiveness. (Provide your plots and explanation in the `.pdf` report file.)

Ultimate Showdown: Tuning Parameters and Comparing Methods

6. (5 points): Optimizing and Evaluating Performance with `compare.py`

   (a) Use the `compare.py` script to experiment with various methods and explore the most effective parameters for each technique we've studied. While it's not necessary to find the absolute best parameters, run multiple tests and select the most suitable ones based on your findings. (Look for `# Modify this param` comment in `compare.py`)

   (b) Run the command `python compare.py`. Note that this process may require some time to complete.

   (c) Once the process finishes, review the results in the file `results/final_comparison.pdf`. Which method would you recommend to aid Aragorn in his quest? (Provide your plots and explanation in the `.pdf` report file.)

2 Upper Confidence Bounds in MABs

Consider the multi-armed bandit problem where at each stage $t = 1, \cdots, T$:

- the learner chooses an arm from a finite set $\mathcal{A}$
- each arm $a \in \mathcal{A}$ generates a random reward $r_t(a) \in [0, 1]$ of mean $\mathbb{E}[r_t(a)] = \mu_a$.

We denote by $A_t \in \mathcal{A}$ the arm chosen at time t. A common way to measure the performance of a bandit algorithm $\mathfrak{A}$ is through *regret*. The regret measures the expected gap between the accumulated reward obtained from "choosing the optimal arm at each step" and the one obtained from "choosing the arm output by algorithm $\mathfrak{A}$". Formally,

$$R_T(\mathfrak{A}) := \max_{a \in \mathcal{A}} \mathbb{E}\left[\sum_{t=1}^{T} r_t(a)\right] - \mathbb{E}\left[\sum_{t=1}^{T} r_t(A_t)\right]$$

Denote the optimal arm by a^* and define as $N_T(a) := \sum_{t=1}^{T} \mathbb{I}\{A_t = a\}$ the number of times arm a has been pulled after T rounds. Note that $N_T(a)$ depends on the realization of the rewards, so it is a random variable.

7. (15 points)

   (a) (5 points) Prove that:

   $$R_T(\mathfrak{A}) = \sum_{a \neq a^*} \mathbb{E}[n_T(a)]\Delta_a,$$

   where for all $a \in \mathcal{A}$, $\Delta_a := \mu_{a^*} - \mu_a$. Given this relation, what suffices to minimize in order to get minimal regret?

   (b) (10 points) Consider the UCB algorithm. It selects an arm at each round according to

   $$A_t := \arg\max_{a \in \mathcal{A}} \text{UCB}_a(t),$$

   where $\text{UCB}_a(t) := Q_t(a) + \sqrt{\frac{2\log(1/\delta)}{N_t(a)}}$, and $Q_t(a)$ is the empirical mean reward of arm a. Remarking that $N_t(a) \leq t$, show that:

   $$\mathbb{P}\left(\mu_{a^*} \geq \min_{t=1,\cdots,T} \text{UCB}_{a^*}(t)\right) \leq T\delta.$$

   Hint: You may use the following fact: For all arms $a \in \mathcal{A}$ and all $\epsilon \geq 0$,

   $$\mathbb{P}(\mu_a \geq Q_t(a) + \epsilon) \leq \exp\left(-\frac{t\epsilon^2}{2}\right).$$