BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
WORK INTEGRATED LEARNING PROGRAMMES DIVISION

Deep Reinforcement Learning
Lab Assignment 1

Total Marks = 12 Marks (6M for the first part and 6M for the second part)

Intended Learning Outcome:
Students should be able to

- Understand the basic functionality of Multi-Armed Bandit and Dynamic Programming.
- Implement the concepts of Multi-Armed Bandit and Dynamic Programming.

Prerequisite:

(1) Students should go through the lectures CS1, CS2, CS3, CS4, CS5 ;
(2) Webinar demonstrations

Please note that this assignment will involve some amount of self-learning (on the part of modelling solutions appropriately + programming skills)

Submission Deadline: 10th January, 2025

Instructions:

- Read the assignment proposal carefully.
- Solve both assignment problems. Submit two different solution files. (Team # - MAB, Team # - DP)
- It is mandatory to **submit** the assignment in **PDF format only** consisting of all the outcomes with each and every iteration. Any other format will not be accepted.
- Add comments and descriptions to every function you are creating or operation you are performing. If not found, then 1 mark will be deducted. There are many assignments that need to be evaluated. By providing the comments and description it will help the evaluator to understand your code quickly and clearly.

How to reach out for any clarifications:
This assignment is administered by

(1) Pooja Harde - pooja.harde@wilp.bits-pilani.ac.in
(2) Divya K - divyak@wilp.bits-pilani.ac.in
(3) Dincy R Arikkat - dincyrarikkat@wilp.bits-pilani.ac.in

Any request for clarification must be addressed through email (official email only) to all the three instructors listed.

Messaging in TEAMS is discouraged. This is to ensure we maintain track of all the transactions. If we find any clarifications to be shared across all the students, we will share this using discussion forums.

Part #1 - MAB

Total Marks = 6 Marks

Title: Optimizing Movie Recommendations Using Multi-Armed Bandits

Background: In the world of online streaming, user satisfaction and engagement are critical metrics for the success of a movie recommendation system. A well-designed recommendation algorithm can significantly enhance the user experience by suggesting movies that align with their preferences, leading to higher platform retention and usage. Recommendation systems face the challenge of balancing exploration (discovering new movies) with exploitation (recommending known favourites) to maximize user satisfaction over time.

Scenario: Imagine a leading online movie streaming platform, TrendMovie Inc., that aims to become the go-to destination for personalized movie recommendations. The platform features a vast collection of movies catering to diverse audiences. TrendMovie Inc. wants to optimize its recommendation strategy to deliver maximum user satisfaction while maintaining a high level of engagement. Each movie recommendation is treated as an interaction with the user, and their feedback is used to refine the recommendation strategy dynamically.

Objective: Your objective is to design and implement a recommendation system using Multi-Armed Bandit (MAB) algorithms to maximize cumulative user satisfaction. The system should dynamically allocate recommendations by learning user preferences in real-time, striking the right balance between exploration and exploitation.

Dataset Description: The dataset contains user ratings for a variety of movies. Key columns in the dataset include:

- **User ID**: A unique identifier for each user.
- **Movie ID**: A unique identifier for each.

- **Rating**: A score provided by the user for a movie (on a scale of 1 to 5).
- **Timestamp**: The time when the rating was given (optional for this assignment).

Link for accessing the dataset:
https://drive.google.com/file/d/1gfobhqlVCw8Oo52JCiYpEBGhG5k7cWBr/view?usp=drive_link

Environment Details:

Arm:

Each movie represents an "arm" in the MAB framework. The probability of a movie being liked by a user is initially unknown and will be estimated based on user feedback during the interactions.

For example:

- **Arm 1**: Movie A
- **Arm 2**: Movie B
- **Arm 3**: Movie C …. and so on, for all movies in the dataset.

Reward Function:

The reward function is defined based on user ratings:

- **Reward = 1**: The user rates the movie high star (e.g., 4 or 5 stars).
- **Reward = 0**: The user rates the movie low star (e.g., 1, 2, or 3 stars).

Assumptions: Run simulations for 1000 iterations for each policy

Requirements and Deliverables:

1. Load the movie ratings dataset and preprocess it as needed. **(0.5 Mark)**
2. Define the environment and compute rewards. **(0.5 Mark)**
3. Implement a random policy for movie recommendations and print each iteration. **(0.5 Mark)**
4. Implement a greedy policy that always recommends the movie with the highest estimated reward and print each iteration. **(1 Mark)**
5. Implement the epsilon-greedy policy, where with probability ε you explore (recommend a random movie) and with probability (1-ε) you exploit (recommend the best-known movie). Try with ε =0.1, 0.2, 0.5, and print each iteration. What value of ε yields the best performance? **(1.5 Mark)**
6. Implement the UCB algorithm for movie recommendations and print each iteration. **(1 Mark)**

7. Plot the cumulative rewards for all policies on a single graph to compare their performance. **(0.5 Mark)**
8. Determine which policy performs the best based on cumulative reward. Provide a concise conclusion (250 words) summarizing the decision-making process and the trade-offs between exploration and exploitation. **(0.5 Mark)**

Colab Template: MAB_Assignment_1.ipynb (*Make a copy of the template. Do not send the edit access request.*)

PART #2 - DP

Total Marks = 6 Marks

Title: Treasure Hunt in the FrozenLake Environment Using Dynamic Programming

Problem Statement: Develop a reinforcement learning agent using dynamic programming to solve the Treasure Hunt problem in a FrozenLake environment. The agent must learn the optimal policy for navigating the lake while avoiding holes and maximizing its treasure collection.

Scenario: A treasure hunter is navigating a slippery 5x5 FrozenLake grid. The objective is to navigate through the lake collecting treasures while avoiding holes and ultimately reaching the exit (goal).

Grid positions on a 5x5 map with tiles labeled as S, F, H, G, T. The state includes the current position of the agent and whether treasures have been collected. The environment consists of several types of tiles:

- Start (S): The initial position of the agent, safe to step.
- Frozen Tiles (F): Frozen surface, safe to step.
- Hole (H): Falling into a hole ends the game immediately (die, end).
- Goal (G): Exit point; reaching here ends the game successfully (safe, end).
- Treasure Tiles (T): Added to the environment. Stepping on these tiles awards +5 reward but does not end the game. After stepping on a treasure tile, it becomes a frozen tile (F).

The agent earns rewards as follows:

- Reaching the goal (G): +10 reward.
- Falling into a hole (H): -10 reward.
- Collecting a treasure (T): +5 reward.
- Stepping on a frozen tile (F): 0 reward.

Objective: The agent must learn the optimal policy π^* using dynamic programming to maximize its cumulative reward while navigating the lake.

State space:

- Current position of the agent (row, column).
- A boolean flag (or equivalent) for whether each treasure has been collected.

Action space: Four possible moves: up, down, left, right

Rewards:

- Goal (G): +10
- Treasure (T): +5 per treasure
- Hole (H): -10
- Frozen tiles (F): 0

Environment Details: Modify the FrozenLake environment in OpenAI Gym to include treasures (T) at certain positions. Inherit the original FrozenLakeEnv and modify the reset and step methods accordingly.
Example grid:

S	F	F	H	T
F	H	F	F	F
F	F	F	T	F
T	F	H	F	F
F	F	F	F	G

Requirements and Deliverables:

1. Create the custom environment by modifying the existing "FrozenLakeNotSlippery-v0" in OpenAI Gym and Implement the dynamic programming using value iteration and policy improvement to learn the optimal policy for the Treasure Hunt problem. **(2 Marks)**
2. Calculate the state-value function (V*) for each state on the map after learning the optimal policy. **(1 Mark)**
3. Compare the agent's performance with and without treasures, discussing the trade-offs in reward maximization. **(1 Mark)**
4. Visualize the agent's direction on the map using the learned policy. **(1 Mark)**
5. Calculate the expected total reward over multiple episodes to evaluate performance. **(1 Mark)**

Colab Template: FrozenLake_using_Dynamic_programming.ipynb (*Make a copy of the template. Do not send the edit access request.*)