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

Deep Reinforcement Learning
Lab Assignment 1

Total Marks = 13 Marks (6M for the first part and 7M 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: 20th December, 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 printed. 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.
- No last moment submissions /email submissions will be accepted. Therefore make sure you will submit the first version of your assignment 2 days before the deadline.

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: Multi-Armed Bandit Algorithms for Profit-Aware Product Recommendation

1. Background

A large e-commerce organization operates a **homepage recommendation slot** that displays *one* product to each visiting user. The company wants to maximize **long-term net profit** from these recommendations.

You are provided with a dataset of **498 user sessions**, each representing how a particular user would respond to each of six products.

Link for accessing dataset: Dataset_Product_Recommendation.csv

The dataset structure is:

UserID, Product1, cost1, Product2, cost2, … , Product6, cost6

Each row corresponds to **one user** and contains the following information:

Product X column

- The value under *Product X* represents the **actual revenue** the system would earn from that specific user if Product X were shown.
- Revenue varies across users because:
    - different users purchase different quantities,
    - some users click while others ignore,

- some users spend more,
- some users purchase premium items or bundles.

Thus, **actual revenue is user-dependent, not product-dependent.**

The same product may generate high revenue from one user and very low revenue from another.

Cost X column

The value under *cost X* represents the **personalized promotional cost** incurred to show Product X to that user. This may include:

- dynamic advertisement bidding cost (CPC),
- discounts or coupon values offered,
- targeted promotion cost,
- personalization overhead.

Costs differ across users because:

- some users require higher incentive levels,
- some are expensive to target via ads,
- some convert easily with minimal cost.

Thus, cost is also user-dependent and varies per session.

2. Environment Dynamics

2.1 Arms

In the Multi-Armed Bandit formulation:

- There are 6 arms:

$A = \{1, 2, 3, 4, 5, 6\}$

- Arm X corresponds to recommending Product X.

2.2 Reward

For each product X and each user session i:

$NetReward_{i,X} = Product_{i,X} - Cost_{i,X}$

This net reward represents the profit gained from showing product X to user i. This is the metric the organization wants to maximize.

3. Current Organizational Policy

The organization currently follows a **random recommendation policy**:

"For each user visit, randomly select one of the six products and display it."

This strategy:

- does not learn from previous outcomes,
- does not identify which products consistently deliver higher profit,
- wastes impressions on low-value products,
- provides unstable and suboptimal profit performance.

Management asks you to investigate:

1. Whether a learning-based approach yields better decisions?
2. Which products are actually profitable when user variability is considered?
3. How do alternative strategies compare to the random policy?

Code Template: MAB_Assignment_1_Code_Template.ipynb

4. Requirements and Deliverables

Q1. Add a Net Reward column that represents the profitability of the six products for all 498 users. Based on this computation, identify one product that frequently gives high positive profit and one product that frequently gives low or negative profit across the user sessions. **[1 Mark]**

Q2. Simulate several rounds (assume 300 rounds or higher) of the organization's existing strategy using the net reward. **[1 Mark]**

- Which product appears good and which appears poor under the existing process?
- What is the approximate average profit per round under this strategy?
- Did randomness occasionally pick a good product?

Q3. Instead of picking randomly, try each product a few times, measure the average profit, and then always select the best-performing one. Using this strategy, determine the cumulative net profit achieved. Which product tends to be repeatedly chosen? **[0.5 Mark]**

Q4. The analytics team suggests improvements to avoid premature lock-in. Instead of relying completely on the first few impressions, occasionally test other products so that we don't miss a better option. **[2.5 Mark]**

- Does very little exploration cause the system to get stuck with a poor product?
- Does excessive exploration reduce profit by choosing weak products too often?
- Which exploration level appears to give the most balanced outcome for this dataset?

Repeat the experiment with three different exploration frequencies, for example:

- Low exploration: 2%
- Moderate exploration: 10%
- High exploration: 25%

Q5. After reviewing results, leadership proposes: *"Testing alternatives is useful, but unnecessary exploration wastes impressions. Explore only when you are uncertain about a product's performance."* Identify a product that was initially tried but later received minimal trials. **[0.5 Mark]**

Q6. Plot the cumulative net profit curves for all strategies tested. **[0.5 Mark]**

- Which strategy produced the highest long-term cumulative profit, and why?
- Which product consistently emerged as the most profitable choice?

PART #2 - DP

Total Marks = 7 Marks

1. Problem Statement

Design and implement a reinforcement learning agent using dynamic programming (value iteration or policy iteration) to compute an optimal policy for a simplified chess game. The agent plays as White and must learn how to convert an advantage into a win or at least avoid a loss in a MiniChess game against a defensive opponent. The problem must be modelled as a finite MDP. Register number of first student in a group (alphabetically sorted) will be considered for configuration design.

The student will:

- Implement a custom Mini Chess environment.
- Use dynamic programming to compute the optimal value function and policy.
- Analyze how state design and reward shaping affect the learned policy and convergence.

2. Scenario

You are building a “Mini Chess Game” for beginner players. The coach focuses on a small, tractable game:

White: King + Pawn
Black: King
Board: 4×4 or 5×5 MiniChess board

White moves first and tries to either:

- Promote the pawn and then deliver checkmate, or
- Force a checkmate directly (if possible)

Black tries to prevent this by blocking the pawn, chasing the white king, or capturing the pawn. The game is restricted to this small set of pieces and a tiny board.

3. Environment Description

Board and Pieces

- Board size:
    - If the student roll number / registration number is even: use a 4×4 board (rows 0–3, cols 0–3).
    - If odd: use a 5×5 board (rows 0–4, cols 0–4).
- Pieces always present:
    - White King (WK)
    - White Pawn (WP)
    - Black King (BK)
- No castling, no en passant, no promotion to anything other than Queen.

Legal Moves

- Kings move like normal chess kings - one square in any direction (8- neighborhood), staying on the board.
- Pawn:
    - Moves one square forward (towards larger row index or smaller row index – the student must choose and clearly document a convention).
    - Captures diagonally forward by one square.
- All usual constraints apply:
    - Kings cannot move into check.
    - Two kings may never occupy adjacent squares (illegal state).
    - A piece cannot move through other pieces.

4. **Episode Termination**

- An episode ends when any of the following happens:
  - Checkmate (White checkmates Black).
  - Stalemate (side to move has no legal moves but is not in check).
  - Pawn Capture (Black captures the White pawn).
  - Pawn Promotion (White pawn reaches last rank and becomes a Queen). After promotion, they may either:
    - (a) terminate immediately with a reward, or
    - (b) continue playing with a Queen replacing the pawn.
  - The student must choose one approach and justify it.
  - Move limit exceeded (e.g., 20 or 30 plies) – draw

a. State Space

Each state should minimally encode:

- Coordinates of WK: (r_wk, c_wk)
- Coordinates of WP (or a special value if promoted/captured): (r_wp, c_wp) or status flag
- Coordinates of BK: (r_bk, c_bk)
- Player to move: {White, Black}
- Any additional flags that can be necessary like,
  - Has the pawn promoted?
  - Check / checkmate / stalemate indicators.

The student must:

- Describe the state representation clearly.

b. Action Space

- For each state, actions are the legal moves for the side to move:
  - Move King to a legal square
  - Move Pawn / promoted Queen

The student must implement a function that, given a state, returns all legal actions.

c. Rewards

The student has to define the reward schemes like:

- Sparse Reward (R)
    - White checkmates Black: +10
    - Pawn gets captured: -10
    - Stalemate or draw by move limit: 0
    - All non-terminal moves: 0

Code Template: DP_Assignment Code Template - ChessGame DP.ipynb

5. Requirements and Deliverables

1. Custom MiniChess Environment [1 Mark]

- Implement a custom environment class with:
    - reset()
    - step(action)
    - render() (text or simple visualization)
- The environment must enforce:
    - Legal moves
    - Check, checkmate, stalemate
    - Pawn moves, captures, and promotion
    - Turn switching (White and Black)

2. Dynamic Programming Solution – Value Iteration and Policy Iteration [2 Marks]

Using the environment as an MDP with reward scheme R,

1. Enumerate all reachable states starting from the initial configuration, which must depend on the ID:
    - If the student ID ends with 0–4:
        - Place WK and BK on opposite corners; pawn on the rank closest to White.
    - If the student ID ends with 5–9:
        - Place WK near the middle; BK on a border; pawn one rank in front of the king (but not protected by it).

    This initial configuration must be explained.
2. Implement value iteration and policy iteration to approximate the optimal value function $V*(s)$ and policy $\pi*(s)$ for White.

3. Stop when the maximum change in value across all states falls below a tolerance (e.g., $\theta = 10^{-3}$).
4. A table or summary of convergence statistics with Number of iterations, Final max value change and Runtime
5. Visual examples of the learned policy from 2–3 different starting states (e.g., printed arrows for moves, sequence of moves shown)

3. State-Value Function Analysis [2 Marks]

1. Visualize the state-value function for All legal positions where it is White to move, the pawn is not yet promoted, and BK is not in checkmate. Since the state space is large, choose any particular state, for example:
    - Fix pawn position + fix black king position, move only white king - visualize this with heatmap for White king at every possible square → value for that state
    - Or fix 'to move = White; pawn is on 2nd rank' and vary both the kings. - visualize this similarly.
2. At least two heat map or diagram showing how $V*(s)$ varies across a family of states for both the techniques.
3. A brief explanation of what structural patterns the student observe (for example, "values are higher when the white king is closer to the pawn and the enemy king is far away").

4. Result Discussion: State Design & DP Limits [2 Marks]

Write an explanation on

1. Curse of Dimensionality:
    - Estimate how the state space would grow if they:
        - Increased board size to 8×8
        - Added more pieces (e.g., a rook or another pawn)
2. Is Dynamic Programming enough for Full Chess?
    - Provide a brief argument for whether DP is tractable for full chess, and how modern RL methods (e.g., Monte Carlo tree search, function approximation) relate conceptually to what is implemented here.