AIL 722: Assignment 3
Semester I, 2025-26

Due: November 2, 2025 12:00 PM

Instructions:

- This assignment has four parts.
- You should submit all your code (including any pre-processing scripts you wrote) and any graphs you might plot.
- Include **a write-up (pdf) file**, which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- Please use Python for implementation using only standard python libraries. Do not use any third-party libraries/implementations for algorithms.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).
- Starter code is available at this link.

1 Deep Q Networks during distribution shift (10 Points)

In this section, you will be working with the cliff environment which was used for Assignment 2, but we have updated the transitions slightly, so please use updated environment given in the starter code.

Create two environment instances:

```
# Training environment
train_env = MultiGoalCliffWalkingEnv(train=True)

# Evaluation environment
eval_env = MultiGoalCliffWalkingEnv(train=False)
```

Tasks which needs to be performed

1. Train two DQN networks (one linear network, one non-linear) on *train_env*. Also, please use standard practices like target network, replay buffer to stabilize training. Also, please add your code in the *'cliff_starter_code.py'*
2. Save the best-performing model in the $Q1/models$ folder for both linear and non-linear models with the names *best_linear.pt*, and *best_nonlinear.pt* respectively.
3. Plot the training rewards for both linear and non-linear models and save it in the $Q1/plots$ folder with filename $cliff\_average\_rewards\_linear.png$ and $cliff\_average\_rewards\_nonlinear.png$.

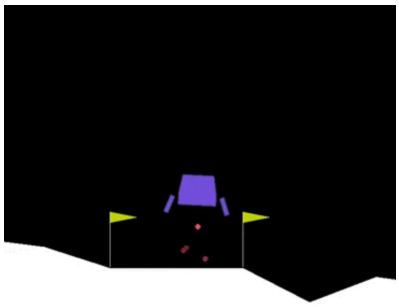

Figure 1: LunarLandar-v2 Environment

4. Using the best saved model, perform evaluation for 100 episodes on *eval_env* for both linear and non-linear models and add mean and std of the rewards in the $Q1/evaluation/cliff\_evaluation\_results.json$
5. Explain your observations about generalization and the bias-variance tradeoff.

2 Deep SARSA (30 Points)

2.1 Problem Description

In this section, you will implement the Deep SARSA algorithm (Neural networks with the sarsa update rule) to solve the LunarLander-v2 environment from Gymnasium. The LunarLander Fig. 1 is a classic control problem where an agent must learn to land a spacecraft safely on a designated landing pad. The environment provides an 8-dimensional continuous state space (position, velocity, angle, angular velocity, and leg contact information) and a discrete action space with 4 possible actions (do nothing, fire left engine, fire main engine, fire right engine).

The goal is to land the spacecraft between two flags with minimal fuel consumption while maintaining stability. Runtime for this question takes around 2 hours on the I9 CPU.

2.2 Tasks

- Implement Deep SARSA using the sarsa update rule. You may use a temporary replay buffer to store transitions from the current policy (discarding them after each update). Optionally, you may use a target network (and optionally you may follow or modify the training structure in Subsec. 2.3). Note that, please implement your code in the *'deep_sarsa.py'* file, after execution, it should create gifs, perform evaluation, and save the models.
- Save the best-performing model in the $Q2/models$ folder with the name *best_deep_sarsa.pt*
- Use the best saved model for the final evaluation for 100 epsiodes and report mean and std of the rewards in the $Q2/evaluation/lunarlandar\_evaluation\_results.json$
- Create the GIF using the best saved model and save it in the $Q2/gifs$ with the name $lunarlandar.gif$

2.3 Training Structure

1. **Data Collection:** Collect a batch of transitions using the current policy π_θ
2. **Training:** Use the collected transitions to update the policy network using sarsa update rule
3. **Evaluation:** Evaluate using the current policy for the 10 episodes
4. **Discard:** Completely discard all collected transitions
5. **Repeat:** Repeat until convergence

3 DQN for Portfolio Management (30 Points)

In this task apply the DQN algorithm to manage the wealth of a portfolio of cash and assets. An initial amount of cash is given at the start of the simulation (50 units). At each time step you can buy or sell the assets by paying a transaction cost and the asset price at that time. You may also choose to forgo the buying and selling and keep the status quo. The prices of the assets change during each time step and can be read from the observations. Cash value remains constant i.e. it does not automatically decay or accrue interest. The wealth of the portfolio is calculated by adding the values of cash and total value of each asset in the portfolio. There is a limit on the quantity of each asset that can be bought/sold at each step (2). The episode ends when a fixed number of time steps have been executed (10).

Details of the environment are given below :
Number of assets besides cash is 5. Initial cash is 50. Step limit is 10. Maximum number of units of each kind of asset that can be held in the portfolio at any time is 10. Each asset can be bought in a lot size of 1 or 2. When selling, the lot size is represented as -1,-2. For keeping the status quo the lot size is 0. Buy/sell costs 1 per transaction. Asset mean represents the mean price of the asset over the episode. Observation is an array of [cash , p1(t), p2(t), ... q1(t), q2(t), ..]
Action is quantity to purchase or buy. Negative quantity implies sell, positive quantity implies buy. For example [1,0,1,1,-1] represents that actions of buying 1 unit each of 1st, 3rd and the 4th asset and selling 1 unit of 5th asset. We limit the environment so that selling assets that are not held will not have any impact. Similarly buying assets without cash will not result in change in state of the asset holdings.

3.0.1 Problem Formulation

- T: investment horizon (number of periods), here $T = 10$.
- N: number of risky assets (excluding cash), here $N = 5$.
- $p_i(t)$: price of asset i at time t, $\quad i = 1, \ldots, N$.
- b_i: buy transaction cost for asset i.
- s_i: sell transaction cost for asset i.
- $H_i^{\max}$: maximum allowable holdings of asset i.
- C_0: initial cash available at $t = 0$.

For each period $t = 0, \ldots, T-1$ and asset $i = 1, \ldots, N$:

- $x_i(t) \in \{-2, -1, 0, 1, 2\}$: number of units of asset i bought ($x_i(t) > 0$) or sold ($x_i(t) < 0$) in period t.
- $h_i(t) \geq 0$: number of units of asset i held at the end of period t.
- $C(t) \geq 0$: amount of cash held at the end of period t.

$$W(t) = C(t) + \sum_{i=1}^{N} p_i(t) \cdot h_i(t), \quad t = 0, \ldots, T.$$

For each $t = 0, \ldots, T-1$:

$$h_i(t+1) = h_i(t) + x_i(t), \quad \forall i$$

$$C(t+1) = C(t) - \sum_{i=1}^{N} x_i^+(t) \cdot (p_i(t) + b_i) + \sum_{i=1}^{N} x_i^-(t) \cdot (p_i(t) - s_i)$$

where

$$x_i^+(t) = \max\{x_i(t), 0\}, \quad x_i^-(t) = -\min\{x_i(t), 0\}.$$

$$h_i(t) \geq 0, \quad \forall i, \ t = 0, \ldots, T \tag{1}$$
$$h_i(t) \leq H_i^{\max}, \quad \forall i, \ t = 0, \ldots, T \tag{2}$$
$$C(t) \geq 0, \quad t = 0, \ldots, T \tag{3}$$
$$x_i(t) \in \{-2, -1, 0, 1, 2\}, \quad \forall i, \ t = 0, \ldots, T-1 \tag{4}$$

$$\max_{\{x_i(t)\}} \quad W(T) = C(T) + \sum_{i=1}^{N} p_i(T) \cdot h_i(T)$$

1. Implement the DQN algorithm to maximize the total value of the portfolio environment. Evaluate the trained model on 100 seeds. Plot the mean portfolio wealth for all ten steps for the 100 seeds. On the same plot also show the standard deviation of the wealth for all 10 steps for the 100 seeds. Report the mean total wealth at the end of 10 steps. Report the model specification and the hyper-parameters. Plot the loss curve with number of optimization steps. Report the ratio of mean and standard-deviation at the final timestep. Submit the code and trained model along with the report.

2. Implement the DQN algorithm to maximize the wealth across all time steps. (The agent should take actions to maximize the wealth for all 10 time steps so that the holder has less risk). Evaluate the trained model on 100 seeds. Plot the mean portfolio wealth for all ten steps for the 100 seeds. On the same plot also show the standard deviation of the wealth for all 10 steps for the 100 seeds. Report the mean total wealth at the end of 10 steps. Report the model specification and the hyper-parameters. Plot the loss curve with number of optimization steps. Report the ratio of mean (over all time steps and seeds) and standard-deviation. Submit the code and trained model along with the report.

Note : You may be asked to run the evaluation code and the model training code if necessary. You are free to choose the neural network architecture, loss function and hyper-parameters. Any approaches that do not involve training DQN will not be assigned credit. Asset prices should only be read using interaction with the environment through observations given out when step or reset functions are called. Please avoid hard coding parameters.
Hint : Or-gym environment is adapted from https://github.com/hubbs5/or-gym. The assets prices are correlated and evolve over time. On Ryzen 7 5800h with GeForce RTX 3060 (laptop) the model training and evaluation takes 5 minutes to run. GPU is not strictly necessary for this task. A small 3 layer neural network is sufficient for good performance. For exploration you may try :

$$\epsilon(t) = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \cdot e^{-t/\tau}$$

$\epsilon(t)$: Exploration rate at time or step t
ϵ_{start} : Initial exploration rate (maximum value)
ϵ_{end} : Final exploration rate (minimum value)
t : Current time step or number of steps elapsed
τ : Decay constant controlling the rate of exponential decay

4 Treasure Hunt-v2 (30 Points)

We will work with the TreasureHunt-v2 environment (Fig. 2) provided in the starter code. This environment is a grid world with 4 available actions corresponding to movement in four directions. The objective is to navigate the ship from any starting position to the fort, which has a fixed location in the top-right corner, while collecting the treasure along the way and avoiding pirates. Additionally, some cells contain land, which acts as a barrier to movement. The positions of pirates, land, and treasures are randomized on a 10x10 grid. The state is represented as a four-channel binary image, $s \in \{0,1\}^{4\times10\times10}$. Each channel corresponds to a specific entity: the positions of

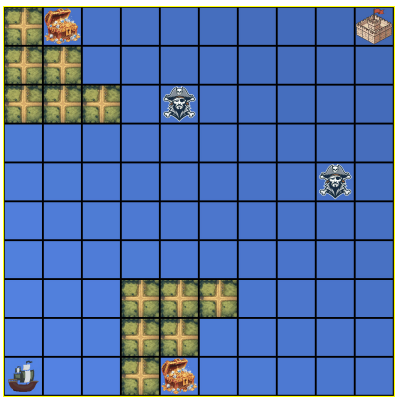

Figure 2: TreasureHunt-v2 Environment

the treasure, ship, land, and pirates are encoded with ones at their respective locations. The expected runtime on HPC A100 GPU is around 7 hours.

1. **Implementation:** Implement an agent that uses a neural network with architecture as shown below, to estimate the Q-function and save your model in the models folder. Please implement your code in the *'dqn.py'* file, after executing, it should save the model, generate plot, save gifs.

Layer Type	**Input Shape**	**Output Shape**	**Kernel Size**	**Stride**	**Padding**	**Activation**
Conv2D	(4, H, W)	(64, H, W)	(3, 3)	1	1	ReLU
Conv2D	(64, H, W)	(64, H/2, W/2)	(3, 3)	2	1	ReLU
Conv2D	(64, H/2, W/2)	(64, H/4, W/4)	(3, 3)	2	1	ReLU
Flatten	(64, H/4, W/4)	(64 * 9)	-	-	-	-
Fully Connected	(64 * 9)	(64)	-	-	-	ReLU
Fully Connected	(64)	(4)	-	-	-	-

Table 1: QNetwork Architecture

2. **Replay Buffer:** Learning a Q-network is known to be unstable. Techniques such as maintaining a replay buffer help stabilize training. Implement a replay buffer to store and sample past transitions for Q-network updates. If the basic replay buffer does not yield satisfactory performance, you are free to modify or extend its design as you see fit.

3. **Exploration Tradeoff**: It is important to maintain a balance between exploration-exploitation during learning. Implement a epsilon decay strategy as below:

$$\epsilon = max(\epsilon_{start} * (Decay\ Factor)^{episode}, \epsilon_{end}) \tag{5}$$

4. **Visualisation**: Using the provided *'./TreasureHunt_starter_code/env.py'* script evaluate the trained agent and save the gif of trajectories of agent with the name *'trained_treasurehunt.gif'*. Compare the trajectory with an untrained/random agent *'random_treasurehunt.gif'* Please save these gifs in the gifs folder.