AIL 722: Assignment 2
Semester I, 2024-25

Due 17th October 2024 11:55 PM

Instructions:

- This assignment has three 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.
- Due to limited support for Windows/MacOS in some Gymnasium environments, we **recommend using Linux-based machines provided by HPC IIT Delhi** (see App. A) for this assignment.
- 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 Model-Based Methods

In this part of the assignment, you will implement the Policy Iteration and Value Iteration algorithms to solve environments with known transitions.

We will work with the TreasureHunt-v1 environment (Fig. 1) provided in the starter code. This environment is a grid world consisting of 400 states, 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. The positions of pirates and treasures remain fixed throughout each episode and are consistent across all episodes. Additionally, some cells contain land, which acts as a barrier to movement. The environment and visualization code is this file.

1.1 Policy Iteration [10 points]

1. **Implementation**: Implement the Policy Iteration algorithm and apply it to the TreasureHunt-v1 environment. Start with a uniform policy.
2. **Visualization**: Visualize the policy on the grid by placing an arrow on each state, indicating the action chosen by the policy at that state. Additionally, generate a GIF of the trajectory obtained by executing the policy. The code for visualizations is provided in the starter code.

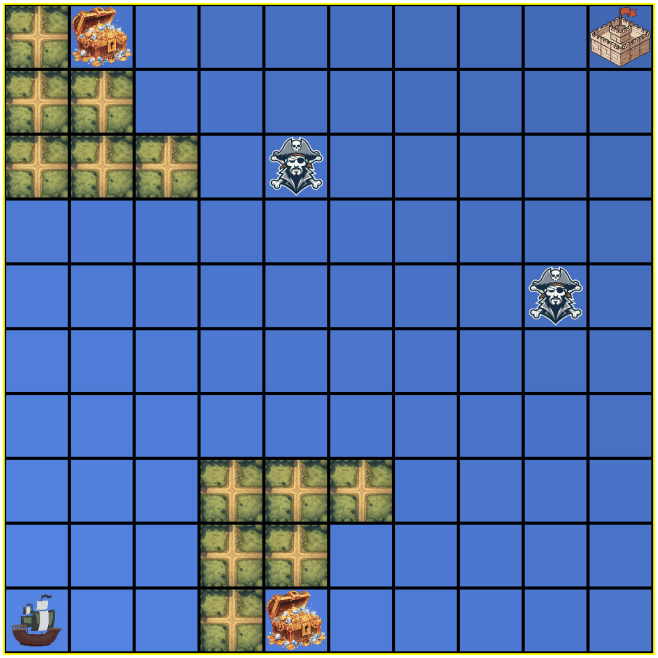

Figure 1: TreasureHunt-v1 Environment

1.2 Value Iteration [10 points]

1. **Implementation**: Implement the Value Iteration algorithm and apply it to the TreasureHunt-v1 environment. Start with a uniform policy.
2. **Visualization**: Visualize the policy on the grid by placing an arrow on each state, indicating the action chosen by the policy at that state. Additionally, generate a GIF of the trajectory obtained by executing the policy. The code for visualizations is provided in the starter code.

2 Model-Free Methods [40 points]

In this part of assignment, you will implement SARSA and Q-learning algorithms to solve environments having unknown transitions (model-free setting).

Table 1 contains the descriptions of the environments that we will work with in this part of the assignment.

Environment	State Space	Action Space	Description
TreasureHunt-v1(Fixed map) [20 points]	Discrete grid(10x10)	4 discrete actions (left, right, up, down)	A grid-based environment where the agent ship must navigate across the sea and reach a goal tile while collecting the treasures and evading pirates on the way.
Taxi-v3[20 points]	500 discrete states	6 discrete actions (north, south, east, west, pick-up, drop-off)	A grid-based environment where the agent controls a taxi, tasked with picking up and dropping off a passenger at specific locations. The agent must avoid obstacles and incorrect actions to minimize the number of steps taken.

Table 1: Environments used for Model Free Methods

The TreasureHunt-v1(Fixed map) and Taxi Environments have discrete state and action spaces. We will solve them using SARSA and Q-Learning.

1. **Implementation:** Implement SARSA, and Q-Learning algorithms (refer S&B Ch 6).

2. **TreasureHunt-v1(Fixed map)** Train the policy above two algorithms. Plot the reward vs episodes. Use a discounting factor(γ) = 0.95 and learning rate(α)=0.3 and probablity of choosing random action $\epsilon = 0.4$. Save the learned policy and Q-table.

3. **Taxi-v3** Train the policy using the above two algorithms. Plot two curves: Reward vs episodes and Epsilon vs episodes curve. Use a discounting factor(γ) = 0.95 and learning rate(α)=0.05. And implement decaying epsilon function as per Eq.1. Save the learned policy and Q-table.

$$\epsilon = max(\epsilon_{start} * (Decay\ Factor)^{episode/K}, \epsilon_{end}) \tag{1}$$

4. **Evaluation and Visualisation**: For both environments using the given *'eval_gym.py'* script evaluate the learned policy by visualizing the trajectories and report the mean reward over 100 trajectories.

5. **Analysis:** Report the number of episodes and cpu time required by each algorithm to get a converged policy on each of the environments.

3 Large-and-Continuous State-Space Environments

Environment	State Space	Action Space	Description
TreasureHunt-v2(Large & Varying)	Discrete of the order 10^{10}	4 discrete actions (Move left, right, top and bottom)	The agent navigates the boat through grid world sea, avoiding pirates and collecting treasures to finally reach the fort. The map gets reconfigured at the start of each episode.
LunarLander-v2	Continuous	4 discrete actions (do nothing, fire left engine, fire main engine, fire right engine)	The agent controls a lunar lander with the objective of landing it on a designated landing pad. The challenge lies in balancing fuel consumption and avoiding crashes.

Table 2: Description of Environments

This part is computationally more intensive than other parts, thus it is advised to start early on this part. The LunarLander-v2 has an expected training time of around 30 mins on a single CPU and the TreasureHunt-v2(Large and Varying) part has an expected training time of around 3 hrs on single CPU.

3.1 LunarLander-v2 [20 points]

This environment is a classic rocket trajectory optimization problem. The environment has discrete actions: engine on or off. The state space is continuous, and thus cannot be represented in a tabular setting. We will use a neural network to learn a state-action value function where the table is implicitly learned in the neural network's weights.

For this part use the provided code template [link] to implement a DQN Agent. The conda environment instructions are available in Sec A.5. The environment does not support Windows/Mac systems, use the HPC resources allotted to you for this part. You need to write your code in *agent.py* and *dqn.py* files. *train.py* and *eval.py* files can be used directly without any change. Store all the hyperparameters in *dqnconfig.yaml*.

1. **Implementation:** Implement an agent that uses a neural network to estimate the Q-function.

   The QNetwork architecture consists of an input layer connected to three fully connected hidden layers. The first hidden layer maps the input state (of size 'state_size') to 32 units, followed by a second layer with 64 units, and a third layer also with 64 units. Each hidden layer uses ReLU activation for non-linearity. Finally, the output layer maps the 64 units to 'action_size' outputs, representing the Q-values for each possible action.

2. **Replay Buffer and Time-Delay:** Learning a Q-network is known to be unstable. Therefore, techniques such as maintaining a replay buffer and employing a time-delayed network as a target are commonly utilized to enhance the stability of training. Implement the Replay Buffer and time-delayed target network for stable training.

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

$$\epsilon = max(\epsilon_{start} * (Decay\ Factor)^{episode/K}, \epsilon_{end}) \tag{2}$$

Train the agent with different values of decay factor as [0.85,0.9,0.95,0.99,0.995,0.999] with ϵ_{start}=1.0 , ϵ_{end}=0.005 and K=100. Plot the reward curves and corresponding epsilon schedules. State your inference and optimum decay factor.

4. **Effect of Wind**: From the env options turn of the wind and retrain the agent, report the reward curve and compare it with previous agents' performance.

5. **Visualisation**: Using the provided *'./LunerLander_starter_code/eval.py'* script evaluate the trained agent and save the video of trajectories of agent. Compare the trajectory with an untrained/random agent.

3.2 TreasureHunt-v2 Environment [20 points]

This environment is an extension of the TreasureHunt-v1 environment. In this version, 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 the treasure, ship, land, and pirates are encoded with ones at their respective locations. The starter code and environment code for this part is in this directory

1. **Implementation:** Implement an agent that uses a neural network with architecture as shown below, to estimate the Q-function.

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 3: QNetwork Architecture

2. **Replay Buffer:** Learning a Q-network is known to be unstable. Techniques such as maintaining a replay buffer stabilize the training. Implement the Replay Buffer.

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

$$\epsilon = max(\epsilon_{start} * (Decay\ Factor)^{episode}, \epsilon_{end}) \tag{3}$$

Train the agent with different values of decay factor as [0.9999, 0.99999] with ϵ_{start}=1.0 , ϵ_{end}=0.005 Plot the reward curves and corresponding epsilon schedules. State your inference and optimum decay factor.

4. **Priority Replay Buffer**: Since positive rewards are very sparse in this environment, a randomly shuffled batch from the replay buffer is likely to contain mostly negative rewards. To facilitate learning in the DQN network for positive rewards, we implement importance sampling from the replay buffer. Specifically, we define a sampling distribution q which ensures equal probability of sampling positive and negative reward experiences. Within each group (positive and negative rewards), all samples have an equal probability of being selected. Additionally, we adjust the loss of the DQN network by applying appropriate weighting to account for the biased sampling..

5. **Visualisation**: Using the provided *'./TreasureHunt_starter_code/env.py'* script evaluate the trained agent and save the video of trajectories of agent. Compare the trajectory with an untrained/random agent.

A Instructions to access HPC Cluster of IIT Delhi

The main website for HPC can be accessed at Supercomputing IITD . A basic overview of hpc is available at Overview PPT. The cluster consists of linux machines, and the basic commands for linux can be found at Linux Commands

A.1 Creating Account

You can apply for HPC account using the following link Apply for account

- Login using your Kerberos ID and Password
- Fill **raunakbh** as 'Faculty supervisor (uid)'
- Choose 15th December as Expiry Date

A.2 Accessing HPC

See Accessing HPC
For Linux and Mac systems HPC can be accessed using Terminal, For windows an SSH Client is required. MobaXterm (Download) is the preffered client.

- From the terminal log in using the command: **ssh kerberos_id@hpc.iitd.ac.in**
- At first login perform one-time setup using the following command:
  source /home/apps/skeleton/oneTimeHPCAccountEnvSetup.sh

The access to the cluster is through a few designated login nodes, which allow users to log in, and not run jobs. For running jobs resources (CPU/GPU) have to be allocated using request commands as mentioned in the section below

A.3 Requesting resources and associated charges

Storage is allocated by default in two portions limited to 100GB of space on /home and 25TB on /scratch directory. /scratch data is not backed up.

The resources are allocated through a Portable Batch System(PBS) that performs job scheduling. See the following webpage for PBS Commands. As an example a script (*'HPC_cpu.pbs'*) is provided in starter code. The usage charges are given at Usage Charges.

For this course, each registered student has been allocated **Rs. 165.0** HPC Credits which is equivalent to around 150 GPU Hours or 1500 CPU-Core Hours through a project named **ail722.{kerberos}.course** which is to be utilized for Assignment 2 & 3.

- Use **-P ail722.{kerberos}.course** as project name to use the allotted credits
- Use **amgr login** to login to projects manager
- Use **amgr checkbalance project -n ail722.{kerberos}.course** to check remaining credits
- Use **qstat -nu $USER** command to check running jobs and **qstat -nTu $USER** command to check estimated start time
- Use **qdel JobID** command to terminate job.

A.4 Accessing Internet and Installing Anaconda

For setting up internet access follow the commands at HPC Internet On the terminal with internet access.

- Use command **wget https://repo.anaconda.com/archive/Anaconda3-2024.06-1-Linux-x86_64.sh** to download Anaconda Installer
- Use **sh Anaconda3-2024.06-1-Linux-x86_64.sh** to install Anaconda

A.5 AIL722 Environment

Run the following commands to setup the environment

- conda create -n ail722 python=3.9
- conda activate ail722
- conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1 cudatoolkit=11.3 -c pytorch
- cd ./DQN
- pip install -r requirements.txt

A.6 Weights and Biases

Weights and Biases (wand) will be used for tracking the runs in this assignment. See the following for a quickstart overview of wandb Quickstart and Colab. The plots and report can be generated effortlessly using the wandb interface, minimizing the need for manual plotting. Commands for using wandb are included in the code template. To run wandb in online mode, make sure the HPC has internet access during non-interactive jobs. Alternatively, you can run wandb in offline mode and later upload the log folder using the **'wandb sync path_to_run_folder'** command.