DS/CMPSC 442: Artificial Intelligence
Fall 2023
Project-3 Reinforcement Learning
(Due: Nov 22nd, 12:20 PM)

Project Description

In this project, every student is required to implement reinforcement learning algorithms learned during the class using **Python 3.9** (this is important, make sure that your code compiles with Python 3.9) to improve the performance of the AI agent under two scenarios. The detail of two questions is mentioned below.

For both two questions, you will need to install **Gymnasium**, an open source Python library (that used to be maintained by Open AI) for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments. The installment of this package is easy, you could directly use the command below:

```
pip install gymnasium
```

Before starting this project, we **strongly recommend everyone check this link** and get familiar with the basic usage of this package (details about how to initialize, interact and modify the environment).

Files to Submit: As part of the project submission, we expect you to submit a zip file which contains only two files. You will need to create and submit 2 files for the assignment by yourselves (See Table 1 below for file name and explanation). ***Make sure that you name these files exactly like this.***

File Name	Explanation
File-1: solution_q1.py	Include all functions that could generate answers for each sub-problem of Q1. Include solutions to all the problems in a single Python file. Make sure that your solution file could be directly run in the terminal (through the command line "python solution_q1.py")
File-2: solution_q2.py	Include all functions that could generate answers for each sub-problem of Q2. Include solutions to all the problems in a single Python file. Make sure that your solution file could be directly run in the terminal (through the command line "python solution_q2.py")

Table 1. Submission files for project-3

Evaluation: Your code will be graded manually by TA for technical correctness. To foster the grading process, they will directly check the output printed in the terminal after running the command "python

solution.py" with different testing version of "input.py" that the instructor has created. So make sure your code can run in the terminal smoothly before making the submission.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy (using automated software). If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Question-1 Solving Blackjack Using Model-Free RL

Blackjack is a card game where the goal is to beat the dealer by obtaining cards that sum to closer to 21 (without going over 21) than the dealers cards. If you have never played Blackjack before, watch this video. In the Gymnasium library, there is a separate environment for Blackjack that we will be using for this question. The description of this environment and corresponding parameters (action space, observation space, and reward rules) can be checked though this link.

Q1.1 (0 marks for grading)

Create the test_q1.py, copy the content below into your test_q1.py, and run test_q1.py to make sure the environment of Blackjack is set correctly.

```
import gymnasium as gym
env = gym.make("Blackjack-v1", render_mode="human") # Initializing environments
observation, info = env.reset()

for _ in range(50):
   action = env.action_space.sample()  # agent policy that uses the observation and info
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

After running test_q1.py, you will see the GUI (which means all packages are set correctly) below:

Note: As you copy the code above, you might have to change quotation characters and other whitespace, which can get corrupted as you copy from MS Word to a Python terminal.

Figure 1. GUI for Blackjack

Q1.2

To win this game, your card sum should be greater than the dealers without exceeding 21. You need to train an AI agent through Q-learning and play optimally with this environment. Please use the environment setting below (solution_q1.py).

```
import gymnasium as gym
env = gym.make('Blackjack-v1', natural=False, sab=False)

###YOUR Q-LEARNING CODE BEGINS



###YOUR Q-LEARNING CODE ENDS
```

Implement a Q-learning algorithm in solution_q1.py, through which you can learn the optimal policy for playing Blackjack in this environment. So we expect you to write code that starts off by initializing the Q(s,a) table for all possible values of s and a. And then you will act according to Q-learning, and every time you act, you will have a chance to update your Q(s,a) table. Over time, if you implement Q-learning properly, you should be able to arrive at the optimal Q(s,a) values, which should enable you to act optimally.

Make sure that you review your slides on Q-learning before you jump into this project.

Questions-2 Solving Frozen Lake Using Model Based RL

The Frozen lake environment on Gymnasium involves crossing a frozen lake from start to goal without falling into any holes by walking over the frozen lake. The player may not always move in the intended direction due to the slippery nature of the frozen lake. The description of this environment and corresponding parameters (action space, observation space, and reward rules) can be checked though this link.

Q2.1 (0 marks for grading)

Create the test_q2.py, copy the content below into your test_q2.py, and run test_q2.py to make sure the environment of Frozen Lake is set correctly.

```
import gymnasium as gym
env = gym.make("FrozenLake-v1", desc=None, map_name="4x4", render_mode="human", is_slippery=True, ) #initialization
observation, info = env.reset()

for _ in range(50):
   action = env.action_space.sample()  # agent policy that uses the observation and info
   observation, reward, terminated, truncated, info = env.step(action)

   if terminated or truncated:
      observation, info = env.reset()

env.close()
```

After running test_q2.py, you will see the GUI (which means all packages are set correctly) below:

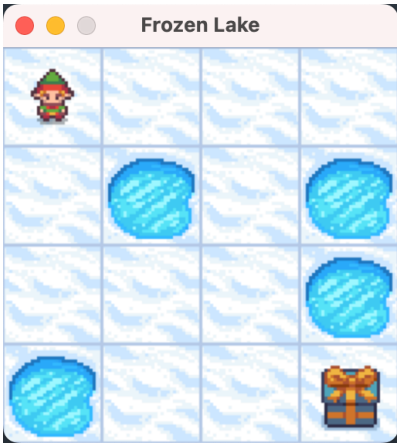

Figure 2. GUI for Frozen Lake

Note: As you copy the code above, you might have to change quotation characters and other whitespace, which can get corrupted as you copy from MS Word to a Python terminal.

Q2.2 Use a random policy to learn the underlying MDP

Write code that will execute a random policy for 1000 episodes, and collect the training data that you observe in those 1000 episodes. You can use the code that is given in Q2.1 as a starting point.

Using the training data for 1000 episodes, estimate the transition function T(s'/s,a) and the reward function R(s,a,s').

Q2.3 Implement value iteration to output the optimal value function

Using the transition function T(s'/s,a) and the reward function R(s,a,s') learned in Q2.2, implement the value iteration algorithm which will help you uncover the optimal value function.

Q2.4 Implement policy extraction

Using the learned value function in Q2.3, implement the policy extraction algorithm which will help you uncover the optimal policy (that is associated with the learned value function of Q2.3).

Q2.5 Write code for acting inside the Frozen Lake policy (which follows the optimal policy you extract)

Using the extracted policy in Q2.4, write code similar to the sample code given in Q2.1 (except for the fact that in Q2.1, the action is chosen randomly at each state, whereas in your code for Q2.5, we expect that each action is chosen according to the optimal policy that was extracted in Q2.4).

At a high level, we expect solution_q2.py to look like the following.

Note: Please make sure that you start off solution_q2.py by using this environment setting.

```
import gymnasium as gym
env = gym.make('FrozenLake-v1', desc=None, map_name="4x4", is_slippery=True)

# Your code for Q2.2 which Executes Random Policy until 1000 episodes


# Your code for Q2.3 which implements Value Iteration

#Your code for Q2.4 which implements Policy Extraction

# Your code for Q2.5 which executes the optimal policy
```

----------END OF PROJECT----------