CS 4122: Reinforcement Learning and Autonomous Systems

Programming Assignment #3 (for Module 3)

Release date: 28th October, 2024, 2:00 pm (IST)

Due date: 12th November, 2024, 11:59 pm (IST)

Maximum Score: 100 marks (this assignment is graded)

Read the following before you move forward:

1. This programming assignment is worth **10% of the total marks** of this course.

2. **Late policy:** Since I am giving only two weeks for this assignment and one of those weeks falls in the fall break, I will be lenient with late submission, i.e. you can submit it late by 3 days (till 15th November) with NO PENALTY. After 3 days, you get ZERO. That said, I will release the 4th programming assignment on 12th November itself. And, 4th programming assignment is significantly more difficult than the 3rd one (which is the simplest till now). So, it is upto you whether you want to submit by 12th November or 15th November.

3. Plagiarism, if detected, no matter how big or small, will lead to a score of ZERO for all the team member; no exceptions! It is ok to take help from AI tools or talk to other teams. But, you cannot copy from another team or any AI tools.
    a. It is down right stupid to copy-paste from AI tools and think that is acceptable.
    b. Also, it is a bad idea to keep open another teams work in front of you while you do the assignment. While you may think you are just taking help, your reports and your codes will get heavily influenced by the other team to the point that it will qualify as being plagiarized.

4. This is a team project. You have to do the project with the team that you selected in the excel sheet that was shared with you.
    a. **Only one submission per team**.
    b. Submission must be made using the google drive link that was shared with you.
    c. **Your submission in the google drive link must contain ONLY SIX files:**
        i. **report.pdf**
        ii. **GymTraffic.py**
        iii. **training.py**
        iv. **policy1.npy**
        v. **policy2.npy**
        vi. **testing.py**

    Read the deliverables carefully to understand what needs to be submitted in the report and in the Python codes. **DON'T submit any other files.** You MUST NOT zip/compress these six files. **You MUST NOT put these six files in a subfolder inside the google drive link.** Directly drop these six files in the google drive folder corresponding to your team.

Traffic Light Control using Reinforcement Learning

This programming assignment has two broad objectives:

1. Building your own custom environments using OpenAI Gymnasium. More specifically, this environment is going to simulate a traffic intersection controlled by a traffic light. Coding your own custom environments is perhaps one of the most important thing that you will learn in this course as far as applied reinforcement learning is concerned. This is because you will get pre-built RL algorithms online that you can use. But, the practical applications where you will be applying these RL algorithms will be very specific to your case. And hence, with high probability, you have to make environments custom designed for your practical application.
2. Implementing variants of temporal difference (TD) based reinforcement learning (RL) agent for the environment mentioned in the first point.

Section 1: Sharing workload

Just like programming assignment 2, sharing workload is a little tricky because the tasks are **not decoupled**. The following are my two cents about sharing workload:

1. Section 4 is about designing the custom OpenAI Gymnasium environment. This is where the **coupling** comes into picture; you can't fully attempt the later sections without completing section 4. So, I suggest that every team member should contribute to section 4. This is also important because of the reason mentioned here.
2. In section 5, there are two tasks that are decoupled. You can easily divide these tasks among yourself and do it independently. Also, you really don't have to wait for section 4 to complete before you can attempt section 5. You can still write a skeleton of the code and then fill in the blanks after section 4 is completed.

Section 2: Useful resources

1. In section 4, you will be **coding a custom environment**. To learn how to code a custom environment, refer to **lecture 28, part 1 (it is a video lecture)**.
2. In sections 5 and 6, you will be training and testing RL agents. More specifically, you will be training RL agents using **modified SARSA** and **triple Q-learning** that you encountered during mock exam and minor 2. For pseudocode of these two algorithms you can refer to the **solution of the mock exam** and **minor 2** in the Dropbox folder.
    a. Python code for Q-Learning is discussed in **lecture 28, part 2 (it is a video lecture)**. The pseudocode of SARSA and Q-Learning is there in lectures 22 to 24.

Section 3: Installation

Installation instructions are same as programming assignment 1. In fact, you don't need any deep learning libraries (like Tensorflow, Keras, and Pytorch) for this programming assignment.

Section 4: Custom OpenAI Gym Environment

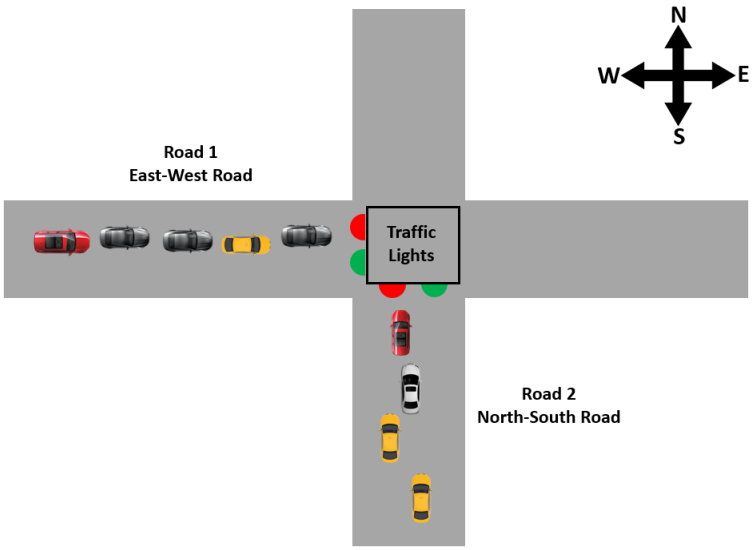

Fig. 1

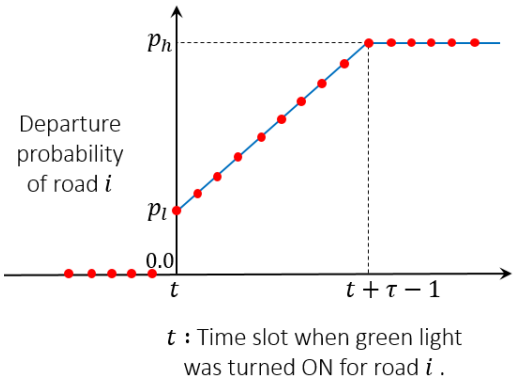

Fig. 2

In this section, your task is to code a custom OpenAI Gymnasium environment for the setup discussed in the following paragraphs.

Consider a traffic intersection as shown in Fig. 1. It consists of an east-west road (road 1) and north-south road (road 2) being controlled by a traffic light. The traffic light can show either green (GO) or red (STOP); there is no yellow light. Obviously, either road 1 or road 2 can be shown a green signal. Also, the traffic light can't show red signal to both the roads at any point of time. The objective of the traffic light controller is to minimize the average wait time of the vehicles in the traffic intersection which can be shown to be equivalent[1] to minimizing the average queue length of the total number of vehicles in roads 1 and 2[2].

We consider that the time is slotted. Let's say that the duration of each time slot is $1\ second$. Vehicles come and join the back of the queue corresponding to each of the two roads. For each of the two roads, at most one vehicle joins the queue in any time slot. The probability that a vehicle will join roads 1 and 2 at any time slot is 0.245 and 0.35 respectively. These are called arrival probabilities.

When the traffic light turns green for a given road, the vehicle starts departing from the front of the queue. At most one vehicle can depart the queue in any time slot. The probability that a car will depart in a time slot is call the departure probability. Now, as soon as the traffic light turns green, the cars can't pick up speed. This is captured in our model by using a departure probability that increases with time as shown in Fig. 2. To elaborate, let's say that at time slot t, the green light for road i turns green. The departure probability of road i at time t is p_l. The departure probability linearly increases from time t to $t+\tau-1$ after which it settles to p_h. For both the roads, $p_l = 0.2$, $p_h = 0.9$, and $\tau = 10$ ($\tau = 10$ means that the vehicle takes about 10 seconds to catch speed which seems reasonable).

---

[1] This equivalence is due to Little's law. You don't have to know about Little's law to do the rest of the assignment.

[2] Even though the objective is to minimize the average queue length, the RL algorithms that you will code in the subsequent sections will minimize the discounted queue length. This is because of simpler convergence guarantee of discounted rewards and something called Blackwell's optimality that I mentioned in module 2. That said, as far as coding the environment is concerned, there will be no change whether we consider average or discounted queue length.

Deliverables:

1. Model this problem as an MDP by documenting the following in **report.pdf**:
    a. **States and state space**. Keep the number of states as small as possible. Otherwise, it is going to slow down training in the subsequent sections.
    b. **Action and action space**.
    c. **Reward** in a time slot and its physical significance. NOTE: I am asking about the reward and NOT the average reward for state-action pairs. Reward is a random variable.
    d. **State transition equations**. HINT: Account for the edge cases; queue length can't be negative.
2. Using the answer to the first question, code a custom OpenAI Gymnasium environment for the setup described in the previous page. The skeleton of this environment is there in **GymTraffic.py** that is provided to you. **GymTraffic.py** contains **GymTrafficEnv** class. Implement the following functions of **GymTrafficEnv** class:
    a. **__init()__**: This function should initialize all the system parameters like the arrival probabilities, p_l, p_h, and τ. You also have to define the observation and the action spaces.
    b. **reset()**: The function to reset the environment in the beginning of every episode. Set the queue lengths of the roads uniformly at random between 0 to 10.
    c. **step()**: This function should take the action as input and return the following:
        i. Next state.
        ii. Current reward.
        iii. $terminated$. This should always be set to $False$ for this setup.
        iv. $truncated$. This setup is a continuous task. But, you must truncate it after 1800 time slots, i.e. each episode is of 30 minutes with the duration of one time slot being 1 second.
        v. $info$. This should be an empty dictionary.

A USEFUL HINT: Queue lengths of the roads will not exceed $\mathbf{1810}$. Think why! This observation is critical because the state space can't be infinite.

Section 5: Training RL algorithms

In this section you will be coding a variant of SARSA and a variant of Q-Learning. You have encountered these variants in minor 2 and the mock exam.

IMPORTANT: In SARSA and Q-Learning (and its variants) we learn Q-values for all state-action pairs. But the state space of the queue length is large for this problem. Because of this the Q-matrix has a dimension of $1811 \times 1811 \times 2 \times 10 \times 2$ which is so huge that it may not even fit in RAM. To resolve this situation, we will NOT learn Q-values for all state-action pairs. Rather, we will learn Q-values for state-action pairs that we are highly likely to encounter. In this section we will assume that a queue length of more than 20 is highly unlikely. Accordingly, we will learn Q-values for all those state-action pairs whose queue length is less than or equal to 20. This will reduce the dimension of the Q-matrix to $21 \times 21 \times 2 \times 10 \times 2$. But, one question remains unanswered: It is still possible that the queue length can become more than 20, however low be the probability. How to decide what action to take in such situations? ANSWER: If queue

length is greater than 20, then we will assume that the queue length is 20 and decide the action accordingly.

Deliverables:

1. Implement triple Q-learning from the mock paper of minor 2 for the custom environment that you coded in the previous section. You must implement this code in the function **TripleQLearning()** of the Python script **training.py** that is provided to you. TripleQLearning() should return the optimal policy (NOT the Q-value) as a Numpy array. Submit the FINAL policy that you learned as **policy1.npy**.

2. Implement the modified version of SARSA from minor 2 for the custom environment that you coded in the previous section. You must implement this code in the function **ModifiedSARSA()** of the Python script **training.py** that is provided to you. ModifiedSARSA() should return the optimal policy (NOT the Q-value) as a Numpy array. Submit the FINAL policy that you learned as **policy2.npy**.
    - If you refer to the pseudocode of modified SARSA in the solution of minor 2, you will see that it is a general code. You have to implement it for $n = 1$ and not the general case.

Note the following:

1. Both **TripleQLearning()** and **ModifiedSARSA()** should learn for 10000 episodes. Use discount factor $\beta = 0.997$ and learning rate $\alpha = 0.1$. It is your task to decide the schedule of the exploration probability.

2. The Python script **training.py** already contains the skeleton of the code where the number of episodes, discount factor, and the learning rate are already set. The function also saves the policy that you learn as .npy files. You MUST NOT TAMPER with this skeleton. Just implement **TripleQLearning()** and **ModifiedSARSA()**.

3. Any additional functions required to implement **TripleQLearning()** and **ModifiedSARSA()** must be declared inside **training.py**.

Section 6: Testing RL algorithms

In this section, you will test the RL algorithms that you trained in the previous section.

Deliverables:

1. You are provided a Python script **testing.py**. Implement the function **TestPolicy()** of **testing.py** that simulates the custom environment that you coded in section 4 for just one episode using the policies that you trained in section 5. After this simulation, the function should plot the queue length of both the roads as a function of time slots. The function should also display the average of the sum of queue lengths of both the roads over one episode.

2. The plots generated in the previous step should be included in **report.pdf**. The report should contain queue length plots for both triple Q-Learning and modified SARSA.