MAE 242: Robot Motion Planning

Project #3

Assigned May 6, due May 27

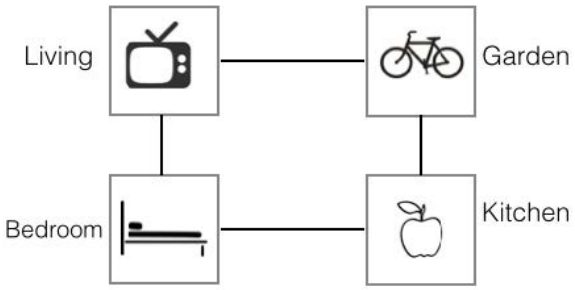
Homework Instructions:

In this third part, you will evaluate the tabular Q-learning algorithms for the Home world game. Recall that the state observable to the player is described in text. Therefore we have to choose a mechanism that maps text descriptions into vector representations.

Here, we will consider a simple approach that assigns a unique index for each text description. In particular, we will build two dictionaries:

- `dict_room_desc` that takes the room description text as the key and returns a unique scalar index,
- `dict_quest_desc` that takes the quest description text as the key and returns a unique scalar index.

For instance, consider an observable state $s = (s_r, s_q)$, where s_r and s_q are the text descriptions for the current room and the current request, respectively. Then $i_r =$ `dict_room_desc`$[s_r]$ gives the scalar index for s_r and $s_q =$ `dict_room_desc`$[s_q]$ gives the scalar index for s_q. That is, the textual state $s = (s_r, s_q)$, is mapped to a tuple $I = (i_r, i_q)$.[1]



Files description: The following files are provided:

- `framework.py` contains various functions for the text-based game environment that have been implemented for you. Some functions that you can call to train and testing your reinforcement learning algorithms:
  - `newGame()`:
    - Args: None

[1] Normally, one builds these dictionaries as the agent is trained collecting descriptions and adding them to the list of known descriptions. For the purpose of this exercise, these dictionaries will be provided to you.

    * Return: A tuple where the first element is a description of the initial room, the second element is a description of the quest for this new game episode, and the last element is a Boolean variable with value False implying that the game is not over.
  - `step_game()`:
    * Args:
      · `current_room_desc`: A description of the current room
      · `current_quest_desc`: A description of the current quest state
      · `action_index`: An integer used to represent the index of the selected action
      · `object_index`: An integer used to indicate the index of the selected object
    * Return: the system next state when the selected command is applied at the current state.
      · `next_room_desc`: The description of the room of the next state
      · `next_quest_desc`: The description of the next quest
      · `reward`: A real valued number representing the one-step reward obtained at this step
      · `terminal`: A boolean valued number indicating whether this episode is over (either quest is finished, or the number of steps reaches the maximum number of steps for each episode).

- `agent_tabular_QL.py`: contains various function templates that you will use to implement your learning algorithm.

You will evaluate your learning algorithm for the Home World game. The metric we use to measure an agent's performance is the cumulative discounted reward obtained per episode averaged over the episodes.

The evaluation procedure is as follows. Each experiment (or run) consists of multiple epochs (the number of epochs is `NUM_EPOCHS`). In each epoch:

1. You first train the agent on `NUM_EPIS_TRAIN` episodes, following an ϵ-greedy policy with ϵ =`TRAINING_EP` and updating the Q values.
2. Then, you have a testing phase of running `NUM_EPIS_TEST` episodes of the game, following an ϵ-greedy policy with ϵ =`TESTING_EP`, which makes the agent choose the best action according to its current Q-values 95% of the time. At the testing phase of each epoch, you will compute the cumulative discounted reward for each episode and then obtain the average reward over the `NUM_EPIS_TEST` episodes.

Finally, at the end of the experiment, you will get a sequence of data (of size `NUM_EPOCHS`) that represents the testing performance at each epoch.

Note that there is randomness in both the training and testing phase. You will run the experiment `NUM_RUNS` times and then compute the averaged reward performance over `NUM_RUNS` experiments.

Most of these operations are handled by the boilerplate code provided in the `agent_tabular_QL.py` file by functions `run`, `run_epoch` and `main`, but you will need to complete the `run_episode` function.

Homework questions: Do the following

1. Write a `run_episode` function that takes a boolean argument (whether the episode is a training episode or not) and runs one episode. *Note:* You should implement this function locally first. Make sure you can achieve reasonable performance on the Home World game before submitting your code.
2. Report performance. In your Q-learning algorithm, initialize Q at zero. Set `NUM_RUNS =10`, `NUM_EPIS_TRAIN` =25, `NUM_EPIS_TEST`=50, ϵ=0.5, `TRAINING_EP`=0.5, `TESTING_EP`=0.05 and the learning rate α=0.1. What is the number of epochs when the learning algorithm converges? (That is, the testing performance become stable)
3. What is the average episodic rewards of your Q-learning algorithm when it converges?

4. (Impact of ϵ on the convergence of Q-learning algorithm). Which of the below do you observe from running the algorithm?

   - For very large ϵ (say $\epsilon = 1$) the algorithm converges slower compared to $\epsilon = 0.5$.
   - For very large ϵ (say $\epsilon = 1$) the algorithm converges faster compared to $\epsilon = 0.5$.
   - For very small ϵ (say $\epsilon = 0.000001$) the algorithm converges slower compared to $\epsilon = 0.5$.
   - For very small ϵ (say $\epsilon = 0.000001$) the algorithm converges faster compared to $\epsilon = 0.5$.

5. (Effects of α). Fix the exploration parameter $\epsilon = 0.5$ and do the experiments with different values of the training $\alpha \in [10^{-6}, 1]$. What you have observed?

   - The algorithm converges for all values of α in less than 200 epochs.
   - The algorithm does not converge for all values of α in less than 200 epochs.
   - The smaller the α the faster the convergence
   - The smaller the α the slower the convergence

Programming Instructions:

- This assigment requires the use of `tqdm, string, csv, sys` libraries.
- This assignment builds upon the functions `tabular_q_learning, epsilon_greedy` that you wrote in `agent_tabular_ql.py`. Now we update the `run_episode` function. You should submit `agent_tabular_ql.py` with your code and comments. Please do not change the other files in this distribution or submit any of our original files other than `agent_tabular_ql.py`.
- The functions modified for this assignment are:

  1. `run_episode`,

  where you provide the run sequence for the RL algorithm. The user defined inputs include discount factor γ, and weight α.

- Your code will be verified for basic technical correctness. Please do not change the names of any provided functions, classes or file names.
- Getting Help:

  1. Some sample code has been provided towards the end of the search file to understand the use of functions. Please start early on the coding assignments. If you are stuck on something, please use office hours and Piazza.

- You are expected to submit a pdf report discussing all outputs and the effect of the parameters on training accuracy.