Audit Access Expires May 25, 2020
You lose all access to this course, including your progress, on May 25, 2020.

4. Tabular Q-learning for Home World game

In this section 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.

In this section you 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

Generating Speech Output

i_q =`dict_quest_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 $I = (i_r, i_q)$.

Normally, we would build these dictionaries as we train our agent, collecting descriptions and adding them to the list of known descriptions. For the purpose of this project, these dictionaries will be provided to you.

Evaluating Tabular Q-learning on Home World

1.0/1 point (graded)
The following python files are provided:

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

In this section, 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 $\varepsilon =$ `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 $\varepsilon =$ `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.

Generating Speech Output

Write a `run_episode` function that takes a boolean argument (whether the epsiode is a training episode or not) and runs one episode.

Reminder: You should implement this function locally first. Make sure you can achieve reasonable performance on the Home World game before submitting your code

Available Functions: You have access to the NumPy python library as `np`, framework methods `framework.newGame()` and `framework.step_game()`, constants `TRAINING_EP` and `TESTING_EP`, `GAMMA`, dictionaries `dict_room_desc` and `dict_quest_desc` and previously implemented functions `epsilon_greedy` and `tabular_QLearning`

```
def run_episode(for_training):
    """ Runs one episode
    If for training, update Q function
    If for testing, computes and return cumulative discounted reward

    Args:
        for_training (bool): True if for training

    Returns:
        None
    """
    epsilon = TRAINING_EP if for_training else TESTING_EP

    epi_reward = 0
    # initialize for each episode
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Generating Speech Output

```
def run_episode(for_training):
    """ Runs one episode
    If for training, update Q function
    If for testing, computes and return cumulative discounted reward

    Args:
        for_training (bool): True if for training

    Returns:
        None
    """
    epsilon = TRAINING_EP if for_training else TESTING_EP
    gamma_step = 1
    epi_reward = 0

    (current_room_desc, current_quest_desc, terminal) = framework.newGame()
    while not terminal:
        # Choose next action and execute
        cur_room_desc_id = dict_room_desc[current_room_desc]
        cur_quest_desc_id = dict_quest_desc[current_quest_desc]
        (action_index, object_index) = epsilon_greedy(cur_room_desc_id,
                                                      cur_quest_desc_id,
                                                      q_func, epsilon)
        (next_room_desc, next_quest_desc, reward,
         terminal) = framework.step_game(current_room_desc, current_quest_desc,
                                         action_index, object_index)

        if for_training:
            # update Q-function.
            next_room_desc_id = dict_room_desc[next_room_desc]
            next_quest_desc_id = dict_quest_desc[next_quest_desc]
            tabular_q_learning(q_func, cur_room_desc_id, cur_quest_desc_id,
                               action_index, object_index, reward,
                               next_room_desc_id, next_quest_desc_id, terminal)

        if not for_training:
            # update reward
            epi_reward = epi_reward + gamma_step * reward
            gamma_step = gamma_step * GAMMA

        # prepare next step
        current_room_desc = next_room_desc
        current_quest_desc = next_quest_desc

    if not training:
```

Generating Speech Output

```
        return epi_reward
```

Test results

See full output

CORRECT

See full output

Submit You have used 3 of 25 attempts

Answers are displayed within the problem

Report performance

1/2 points (graded)

In your Q-learning algorithm, initialize Q at zero. Set `NUM_RUNS` $= 10$, `NUM_EPIS_TRAIN` $= 25$, `NUM_EPIS_TEST` $= 50$, $\gamma = 0.5$, `TRAINING_EP` $= 0.5$, `TESTING_EP` $= 0.05$ and the learning rate $\alpha = 0.1$.

Please enter the number of epochs when the learning algorithm converges. That is, the testing performance become stable.

25 ✖ **Answer:** 15

Please enter the *average episodic rewards* of your Q-learning algorithm when it converges.

0.53 ✔ **Answer:** 0.52

Submit You have used 5 of 6 attempts

Generating Speech Output

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Unit 5 Reinforcement Learning (2 weeks) :Project 5: Text-Based Game / 4. Tabular Q-learning for Home World game

Add a Post

Show all posts by recent activity

- High level pseudocode of the provided functions in agent_tabular_ql.py — 1
  - main(): - load game constants - for NUM_RUNS: - run() --append--> epoch_rewards_test (list ...
  - Pinned Community TA
- [STAFF] Typo error on function tabular_QLearning — 2
  - In the available functions section the previously tabular_q_learning function implemented on ...
- Issue with running main file — 6
  - I get the following error whenever I run the agent_tabular_ql.py script: line 77, in tabular_q_le...
- Why positive epsilon for testing? — 1
  - Why are we having epsilon greater than 0 for testing. For training it makes sense, but should ...
- Numbers in the Epoch Run Progress — 2
- for whoever having trouble with convergence — 3
  - Beware to retrieve the index of the current state inside the while loop
- [Staff] Nudge re my submission? — 2
  - Can I get a nudge pointing me in the right direction? My submission for parts Tabular-Q perfo...
- Report performance — 6 new
  - Hello! I'm not sure about the report performance question. what exactly does it mean: > *the...
- Help with episode reward — 5 new
  - Hopefully, someone can help me see what I am doing wrong. I am doing ok on the training p...
- Evaluating Tabular Q-learning on Home World — 6 new — 10
  - how to update epi_reward? any hints please

Generating Speech Output

Report performance - how many decimal places? 6
I can't get the answer to the average episodic rewards part, even though I have the number o...

Answer marked as correct but graph does not converge 3

? Not sure what's wrong with the way I'm computing the epi_reward...