PSet3 - CS 7648

Interactive Robot Learning, Spring 2024
Instructor: Matthew Gombolay

Due Date: Feb 9th 23.59 Eastern Time

Instructions:

- This problem set has two sections I (coding) & II (short answers) to be submitted separately under two *assignments* on Gradescope.
- For submitting code, please use the gradescope assignment with the suffix '(Code)'. Detailed instructions are below.
- For submitting short answers, please use the gradescope assignment with the suffix '(Short Answers)'. Detailed instructions are below.
- You may work with one or more classmates on this assignment. However, all work must be your own, original work (i.e., no copy + pasting code). You must list all the people you worked with and sources. There will be a dedicated question on the gradescope assignment for short answers.
- **Use of ChatGPT or other AI based resources is prohibited.**

Section 1: Coding

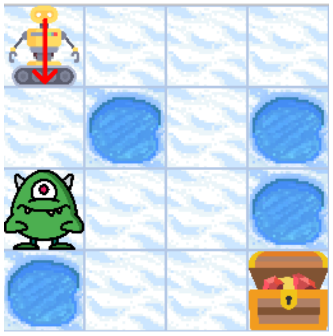
Section 1.0: Setup

Please follow instructions from the previous PSet to setup the environment.

Section 1.1: TAMER

Please refer to this version of the paper: Knox, W. B., & Stone, P. (2009, September). Interactively shaping agents via human reinforcement: The TAMER framework. In Proceedings of the fifth international conference on Knowledge capture (pp. 9-16). https://users.cs.utah.edu/~dsbrown/readings/tamer.pdf

Please **DO NOT** refer to alternative versions of the same paper such as: Knox, W. B., & Stone, P. (2008, August). Tamer: Training an agent manually via evaluative reinforcement. In 2008 7th IEEE international conference on development and learning (pp. 292-297). IEEE.



We would like you to provide feedback and train a robot agent using TAMER (Training an Agent Manually via Evaluative Reinforcement). Please refer to the paper here. For this subsection (1.1), we are implementing Algorithm 1 of the paper. We will use the same grid world domain where the robot is tasked to reach treasure while escaping a monster. For simplicity, the location of the monster is fixed. It is your task to provide the agent expert feedback through TAMER such that the agent can reach the treasure.

Run the following command to start the training process:

```
$ python train_tamer.py
```

The script initializes the environment, initializes the weights of a reward model of the human, begins exploring the environment. As the robot moves around in the environment, you are required to evaluate its actions and provide it with a score.

The reward keys are ‘p’, ‘h’ and ‘o’ which correspond to scores [-10, 0, 10]. If you do not press any key for 5 seconds, the wait for input will terminate, and the code will assume you have provided feedback ‘0’, equivalent to pressing ‘h’. Note that the arrow drawn on the robot indicates the last action taken. **You are required to evaluate the action that has already been taken in the environment**.

The reward model will be updated when you provide nonzero feedback. For the script to run correctly, you are required to fill in code in file **‘tamer_updates.py**’ for functions get_greedy_action(), and update_reward_model() --> we have marked with ‘**# INSERT CODE HERE’**.

As it learns from your scores, it should be able to eventually reach the goal. The dataset and the reward model are saved in pickle files ‘**trained_tamer_std.pkl’** and ‘**dataset_tamer_std.pkl’**. Note that the files are overwritten when you run the script each time.

Your policy requires a minimum mean episode return of **30.0**. You can check the performance of your agent with the following command. Note that the evaluation code will not run unless you at least completely implement the function ‘**get_greedy_action**’.

```
$ python eval.py <reward_model_file_name> <render>
```

Notes:

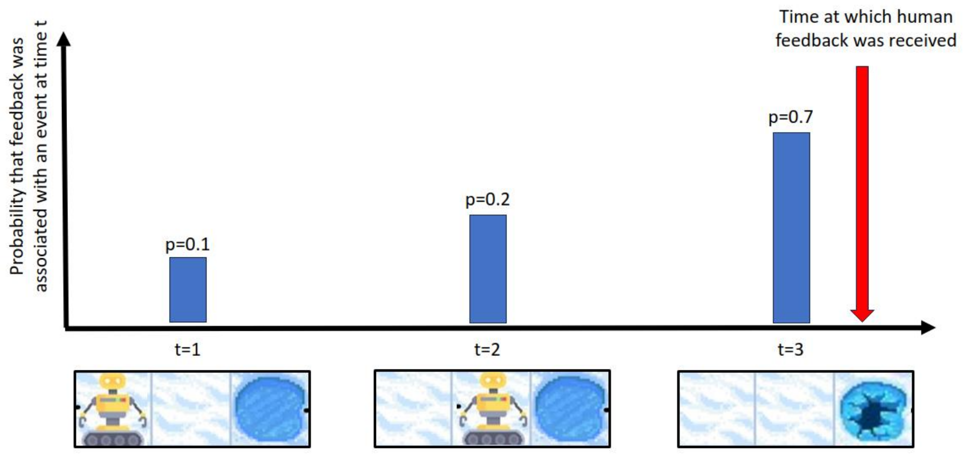
- You are to use the provided function ‘**get_feat_frozenlake**’ to implement the function ‘getFeatures’ in Algorithm 1 of the linked paper. It is already imported into ‘tamer_updates.py’.
- It might not be as easy as giving a demonstration or corrective demonstrations directly to the model. Consistency in your feedback will help.

Section 1.2: Credit Assignment

In this section, we will be implementing Algorithm 2 of the paper. The agent will step through the environment at a much faster pace. Thus, the provided feedback needs to be assigned to the appropriate events (state-action pairs) that happen in the past as there can be delay in providing feedback. To start training, run the following command:

```
$ python train_tamer_async.py
```

The script waits for a shorter time each step for feedback like the one above, and to account for delayed rewards, assigns the feedback to events in the past with certain probabilities. One such probability distribution is a modified gamma function shown in Figure 3 in the paper. To account for discrete time and simplify matters in our implementation, we use a discrete probability distribution over the last K steps (where K=3). Check the figure below:



You are required to implement the function **update_reward_model_with_credit** in '**tamer_updates.py**' at locations marked with '**# INSERT CODE HERE**'.

As it learns from your scores, it should be able to eventually reach the goal. The dataset and the reward model are saved in pickle files '**trained_tamer_async.pkl**' and '**dataset_tamer_async.pkl**'. Note that the files are overwritten when you run the script each time. Your policy requires a minimum mean episode return of **30.0**.

Hints:

- Try to avoid giving unnecessary feedback if the agent performs as you desire. If the agent happens to oscillate between two states, try to either provide negative feedback, changing the default credit weights or restarting the entire training process. Consistency in your feedback will help.

Submission Instructions

- **Submit the files indicated below in a zip. The name of the zip does not matter.**
- **When your `*submitted_code.zip*` is unzipped using the command `*unzip submitted_code.zip*`, it should unzip the contents in the current directory after which your current directory should have the following structure.**
- **You are not required to submit any other files. Any deviation may result in the autograder failing.**

```
submitted_code.zip
├── trained_tamer_std.pkl
├── dataset_tamer_std.pkl
├── trained_tamer_async.pkl
├── dataset_tamer_async.pkl
└── tamer_updates.py
```

Section 2: Short Answers

Please respond to the following questions in short answer form. **Answer directly on the gradescope quiz in the boxes designated to each question.**

1) What are some of the techniques we could use to handle suboptimality in demonstrations? Please list two of them and briefly explain your solutions. Please provide a general answer, not necessarily related to the TAMER framework.

2) In a few sentences each, please compare TAMER with Behavior Cloning (PSet1) and Inverse Reinforcement Learning (PSet2). Please list the pros and cons for each method.

3) Explain credit assignment in TAMER. How would you design the probability distribution for credit assignment in TAMER?

4) In a few sentences, compare TAMER and COACH

Grading:

You will obtain up to 20 points for the short answer section and 80 points for your code implementation. The autograder will only account for a portion. The instructors will review the assignment to provide the final grade. For a detailed rubric, please check gradescope.