[CSCI-GA 3033-090] Special Topics: Deep Reinforcement Learning

Homework - 2

Link for Submission - https://forms.gle/zpjFfATnS7kQhPwR7

Note for Submission :

Submit Code for both the problems in *Link to code* section of the form
Submit PDF for both the problems in *Link to PDF* section of the form
For more information on what to submit, see the text in red below each problem

Deadline for submission**: October 5, 2020**

Imitation Learning

In Part 1 of this HW, we will experiment with imitation learning in the car-racing environment. You will implement the behavior cloning agent and evaluate the performance.

Car-Racing Environment: https://gym.openai.com/envs/CarRacing-v0/

Section 1.1 Getting Started

The starter code repository can be found here:
https://drive.google.com/drive/folders/1p3IJiZA-ApeWYTM11TAPKxjyUpFDi14n?usp=sharing

Install OpenAI gym and the dependencies.

Section 1.2 Behavioral Cloning

1. Familiar yourself with the CarRacing environment by running drive_manually.py in the starter code repository. Car controls are keys -> up, down, left, right. Please go through the code in the repository and see how they are interfacing with each other.

2. Collect the driving data with drive_manually.py --collect_data. This data will be stored in the data folder every 5000 steps. The more you collect the data, the better your agent will get.
This script will create folders /results and /data with your expert score stored in a JSON file.

3. Take the collected data to train the behavioral cloning agent.

4. Improve your agent.

What to submit for Part 1: Imitation Learning?

1. Make necessary changes to the starter code to make it work for Behaviour cloning. Submit code as a zip file.
2. Attach images of frames/data that you collected in 1 page of your PDF submission.
3. Attach images of frames/data that your behavior cloned policy produces in 1 page of your PDF submission.
4. Attach a curve that shows how important is the number of datapoints for behavior cloning in your PDF submission. In this curve, x-axis is the number of training examples and y-axis is the performance of the behavior cloning policy.
5. Additionally, attach screenshots, logs along with anything you feel is necessary as proof that your code worked in your PDF submission.
6. Write up any tricks you had to use to make behavior cloning work better.
7. Extra credit: Implement DAGGER and show how it can obtain better results than behavior cloning.

Value Iteration

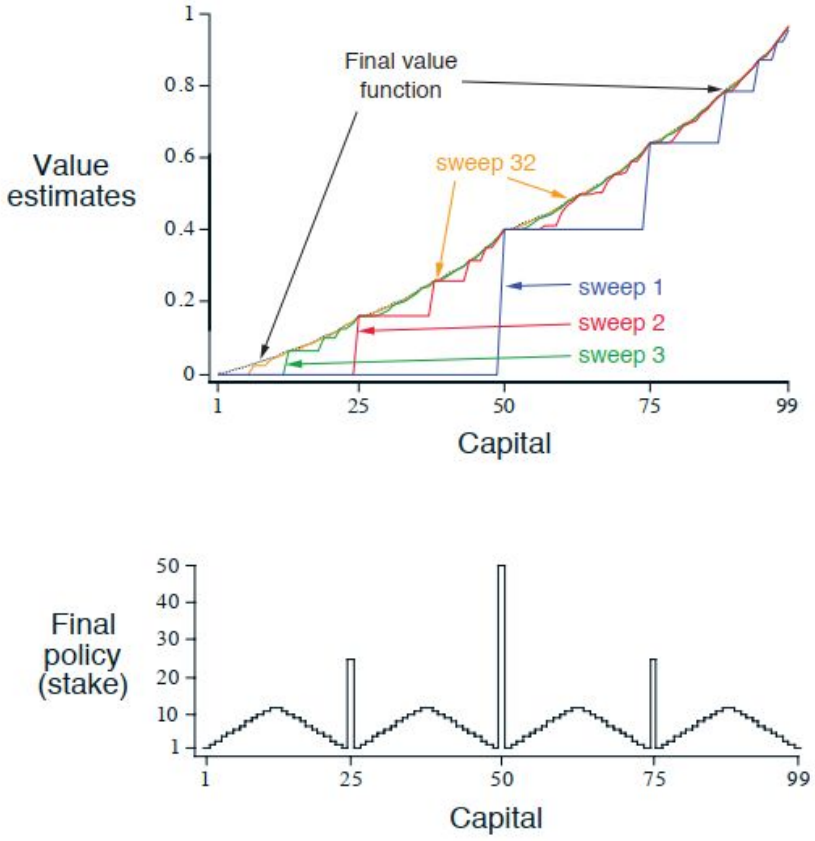

Figure: *The solution to the gambler's problem for ph = 0.4. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.*

(This problem is from Ch 4 of Reinforcement Learning by Sutton)

Section 2.1 Gambler's Problem

Consider a scenario where a gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of $100, or loses by running out of money. On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler's capital, s 2 {1, 2, . . . , 99} and the actions are stakes, a 2 {0, 1, . . . , min(s, 100 - s)}.
The reward is zero on all transitions except those on which the gambler reaches his goal when it is +1. The state-value function then gives the probability of winning from each state. A policy is a mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal. Let ph denote the probability of the coin coming up heads. If ph is known, then the entire problem is known and it can be solved, for instance, by value iteration.

The figure shows the change in the value function over successive sweeps of value iteration, and the final policy found, for the case of ph = 0.4. This policy is optimal, but not unique. In fact, there is a whole family of optimal policies, all corresponding to ties for the argmax action selection with respect to the optimal value function. Can you guess what the entire family looks like?

Section 2.2

Why does the optimal policy for the gambler's problem have such a curious form? In particular, for the capital of 50, it bets it all on one flip, but for the capital of 51, it does not. Why is this a good policy?

Section 2.3 Programming

Implement value iteration for the gambler's problem and solve it for ph = 0.25 and ph = 0.55. In programming, you may find it convenient to introduce two dummy states corresponding to termination with a capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically, as in the figure. Are your results stable as $\Theta \rightarrow 0$?

Section 2.4

What is the analog of the value iteration update for action values, $q_{k+1}(s, a)$?

What to Submit for Part 2: Gambler's Problem?

1. Attach your answers to Section 2 - Gambler's Problem in the PDF.
2. Attach your code for 2.3 in the code zip file.