Assignment 2: Reinforcement Learning

Alban Grastien

2020

This assignment focusses on reinforcement learning (RL) techniques. You will need to implement some RL methods and explain the behaviour of the different algorithms.

1 General Instructions

1.1 List of Files

Domain files These files are used to represent problems on which the algorithm can be applied.

`MDP.py` is the file from the previous assignment. Remember that RL is applied over systems that are generally assumed to be MDPs.

`domain.py` defines the `RLEnvironment` interface which specifies how one can interact with an environment, through the following methods:

- `reset()`: returns the environment to its initial state;
- `applicable_actions()`: returns the list of actions applicable in the current state;
- `execute(action)`: executes the specified action, which returns the new state and the immediate reward;
- `current_state()`: returns the current state.

Notice that, as opposed to an MDP, an `RLEnvironment` does not provide the probability distribution associated with each action. You can only find out about the next states and benefits of actions by interacting with the environment.

`cricket.py` defines a domain that mimics the game of cricket. `cheatingcricket.py` provides a variant of Cricket in which agents have access to a "cheating" strategy. These domains are described in Subsection 1.2.

Files to Implement The following files contain the methods you will need to implement.

`q.py` contains classes and methods related to Q functions, where a Q function associates a value to each pair state/action. A `Q` object includes two methods:

- `value(state,action)` returns the estimated value of the specified action in the specified state. The reward here is the long term reward, i.e., not just the immediate reward from performing the action but also the discounted reward that the agent will obtain in the future.
- `learn(state,action,reward)` tells the Q function the reward associated with the state/action pair the last time this action was performed. From this information, the `Q` object can refine its estimate of the value of the state/action pair. Once again, the reward here is the long term reward, i.e., not just the immediate reward from performing the action but also the discounted reward that the agent will obtain in the future.

The file contains an implementation `LastValueQ`, in which $LastValueQ(s, a)$ is the value of a in state s from the last time that action was executed. (This is a very bad idea in a stochastic context.) This class is here for testing and to give an example implementation.

`sarsa.py` and `qlearning.py` contain an implementation of the two algorithms, SARSA and QLEARNING.

Test Files The files starting with `test_` are used to test the implementations.

The files starting with `question` are used to demonstrate the behaviours of different algorithms. These files use `rewardwrapper.py`, which contains a wrapper that collects the rewards of the implementations, and `plot.py`, which provides a method to draw the rewards.

1.2 Description of the Cricket Domain

The implementations are tested with the CRICKET domain.[1] In this domain, the agent plays the two batters.

The ball is first batted far from the wickets, and it will return to the wicket in the next steps. The two batters can run to exchange their position; they can stop running at any time. If the players are at their starting positions when the ball returns, the players score as many points as the number of position exchanges they managed to perform. If at least one of them is not back at their starting position, the two players get a penalty of -10. The stochasticity comes from the varying speed of the batters and the ball.

The CHEATINGCRICKET domain extends the CRICKET domain with the following action. At the beginning of the run, the batters can decide to *cheat*. If they do cheat, they have to conceal this action by performing the *hide* action before scoring. If they do so, their score is double; if they don't, they get a huge penalty (-100).

Together with the domains are given several instantiations with different parameters.

2 Implementation

This section describes the 6 methods/classes that you are supposed to implement. These methods are generally no longer than a dozen lines.

2.1 greedy_action(q: Q, env: RLEnvironment) -> Action

In file `q.py`, implement the method that computes the greedy action that should be performed in the environment according to the specified Q function.

Test this method with file `test_greedy.py`.

2.2 epsilon_greedy_action(q: Q, env: RLEnvironment, epsilon: float) -> Action

In file `q.py`, implement the method that performs the epsilon-greedy strategy discussed during the lectures.

Test this method with file `test_eps.py`.

2.3 TemporalDifference

In file `q.py`, implement the class `TemporalDifference` that learns using the TemporalDifference strategy with a fixed alpha parameter.

Test this method with file `test_td.py`.

[1] With apologies to the fans of cricket.

2.4 Average

In file `q.py`, implement the class `Average` that stores the average of the learnt rewards given to state/action pairs.

Test this method with file `test_ave.py`.

2.5 SARSA

In `sarsa.py`, implement the method `execute_one_action` which performs one iteration of the SARSA algorithm.

Test this method with file `test_sarsa.py`.

2.6 QLearning

In `qlearning.py`, implement the method `execute_one_action` which performs one iteration of the QLearning algorithm.

Test this method with file `test_qlearning.py`.

3 Interpretation

In this section, you will be asked to explain the behaviours of various algorithms. Specifically, you will have to look at the evolution of the reward function over time, and to explain (in a few sentences) why the rewards associated with one implementation is higher than the other at some points during the execution.

Each python file `question`x`.py` contains a question. Answer this question in a few sentences.

There are two ways to run these files:

1. You can run them directly such as:

   ```
   python3 question1.py
   ```

   and they will use your implementation from the previous section. Notice however that the code may require a substantial amount of time to complete.

2. You can run them using the recorded values:

   ```
   python3 question1.py r solutions/
   ```

   This allows you to answer the questions even if your implementation is incorrect.

You can also use `python3 question1.py w prefix` if you want your execution to be saved (this creates text files such as `prefixq1_1`), and later use the same command with `r` instead of `w` to show the graph again (without re-running the code).

Running the python file will generate a graph that shows how the reward function evolves over time, as well as a table that shows the general trend. If the graph is too messy to interpret (which should happen sometimes), use the table instead.

3.1 Question 1

Explain why Q-learning with $\varepsilon = 0.01$ performs better than $\varepsilon = 0.1$.

3.2 Question 2

Explain why SARSA performs better than Q-learning during the learning process.

3.3 Question 3

Explain why Q-learning performs better than SARSA after the learning process.