Reinforcement Learning: Coursework 2

Pavlos Andreadis

March 2018

Release date: Wednesday 7th March 2018
Due date: 16:00 Monday 2nd April 2018

Introduction

This coursework is concerned with learning an optimal policy for *any instance* of the "Road Fighter" problem, as defined in `https://github.com/cortu01/rl_roadFighter`. It builds on the material covered in the lectures on Markov Decision Processes (MDPs), Monte Carlo and Temporal-Difference Learning solutions to MDPs, and Generalisation and Function Approximation. The aim of the coursework is to better familiarise you with *function approximation* and *sampling* in the context of Reinforcement Learning (RL).

Code & Report

The code for the exercises should be implemented in Matlab, and make use of the code available in the course repository: `https://github.com/cortu01/rl_roadFighter`. Specifically, the exercises ask you to use certain scripts that will define the MDP problem, and any starting policy, that needs to be used. The repository contains brief instructions on getting started with the code.

Where an implementation does not work correctly, comments on the code will be taken into account positively. Submit any Matlab files you have written for the exercises, as well as a local version of the repository code, including any files you might have modified. The solution for each exercise should be executable by running a script with the name `cw2_solution#.m`,where `#` should be replaced by the exercise number. (There is therefore no need to save and submit the results of running the scripts).

Please make sure you have pulled the latest version of the code from the repository.

Exercise 1: Policy Evaluation ||70/100 marks||

Consider the "Road Fighter" problem where you are not given a predefined map, but rather experience a new map at each episode. Furthermore, assume that states cannot be uniquely identified (for example by an id or $(row, column)$ coordinate) but are instead described by a vector of features. These features will correspond to the rewards one would receive if arriving at each of the states visible in the rows in front of the car (replacing these feature values with 0 as you approach the absorbing states at the edge of the map). This would make for a total of 4×5 features (4 rows, 5 columns).

Write a policy evaluation procedure (TD-Learning or Monte Carlo) using such a description of states, and use it to evaluate the policy *implicitly* defined by the state-action (Q) function saved in variable `Q_test1` as produced in the script `cw2_exercise1.m`, for randomly generated maps as generated by the function `generateMap`.

It is important to understand, that the simulation will still run over an explicit representation of states. However, you should use `getStateFeatures.m` , in order to get the feature description of a given state identified by its id-number or $(row, column)$ coordinates. Moreover, you will have to learn a different set of parameters for each action, for a total of $3 \times 4 \times 5$ parameters. Lastly, consider that every sampled episode needs to be drawn from a new randomly generated map.

(bonus) ||20/100 marks|| :

Implement both TD-Learning and Monte Carlo solutions. I should be able to switch between one and the other by changing the assignment to a variable named *ALGORITHM*: 0 for Monte Carlo, 1 for TD-Learning.

Exercise 2: Implementing Control Algorithm ||30/100 marks||

Implement any RL *control* algorithm and use it to find an optimal policy for the scenario in Exercise 1 (using the described function approximation).

(bonus) ||20/100 marks|| :

You are forced to implement a control procedure for this scenario while having access to limited memory. Design your own 3 features computed from the original features outputted from `getStateFeatures.m` , and adapt your code to run with either the original representation or your own. I should be able to switch between representations by changing the assignment to a variable named *FEATURES*: 0 for Original, 1 for Custom.

Mechanics

Marks: This assignment will be assessed out of 100 marks and forms 10% of your final grade for the course. (Anything above 100 will be given a 100/100 score).
Academic conduct: Assessed work is subject to University regulations on academic conduct:
`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`
Submission: You can submit more than once up until the submission deadline. All submissions are time-stamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline. If you submit anything before the deadline, you may not resubmit afterward. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit exactly once after the deadline, and a late penalty will be applied to this submission unless you have received an approved extension. Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same time-frame as for on-time submissions.
Warning: Unfortunately the submit command will technically allow you to submit late even if you submitted before the deadline (i.e. it does not enforce the above policy). Don't do this! We will mark the version that we retrieve just after the deadline, and (even worse) you may still be penalized for submitting late because the time-stamp will update.

For additional information about late penalties and extension requests, see the School web page below. Do not email any course staff directly about extension requests; you must follow the instructions on the web page: `http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`
Late submission penalty: Following the University guidelines, late coursework submitted without an authorised extension will be recorded as late and the following penalties will apply: 5 percentage points will be deducted for every calendar day or part thereof it is late, up to a maximum of 7 calendar days. After this time a mark of zero will be recorded.

Submission

Your coursework submission should be done electronically using the submit command available on DICE machines. Your submission should include

- any comments you would like me to read in file `cw2_report.txt;`
- the script to run your solution for each exercise `cw2_solution#.m,`where `#` should be replaced by the exercise number;
- any other Matlab files you wrote for your solution to the exercises;

- and a local version of the repository code including any changes you made to the files.

You should copy all of the files to a single directory, `coursework2`, and then submit this directory using

```
submit rl cw2 coursework2
```

The submit command will prompt you with the details of the submission including the name of the files / directories you are submitting and the name of the course and exercise you are submitting for and ask you to check if these details are correct. You should check these carefully and reply y to submit if you are sure the files are correct and n otherwise. You can amend an existing submission by rerunning the submit command any time up to the deadline. It is therefore a good idea (particularly if this is your first time using the DICE submit mechanism) to do an initial run of the submit command early on and then rerun the command if you make any further updates to your submission rather than leaving submission to the last minute.