Reinforcement Learning: Coursework 1

Pavlos Andreadis

February 2018

Release date: Wednesday 7th February 2018
Due date: 16:00 Monday 5nd March 2018

Introduction

This coursework is concerned with learning an optimal policy for a specific scenario on 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) and Dynamic Programming solutions to MDPs. The aim of the coursework is to better familiarise you with representations of finite MDPs, as well as with how Policy Iteration works.

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.

The submission does not require a printed document. However, add answers to questions in the exercises to a file named `report.txt`. 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 `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: Implementing Policy Evaluation ||50/100 marks||

Implement the *policy evaluation* algorithm for the "Road Fighter" problem. Use it to evaluate the policy defined by variable `pi_test1` as produced in the script `exercise1.m`, for the specific scenario defined by the object `MDP_1`.
(**Note:** Objects created from the `GridMap` class now have the functions `getTransitions` and `getReward` acting as the transition and reward functions respectively. States are represented as `[row,`column] coordinates or as a state number).

Exercise 2: Questions ||50/100 marks||

Part 1: Assume you have evaluated a policy, for example the policy evaluated in Exercise 1. What procedure could you then run to produce a better policy? Given your answer, can this new policy ever be stochastic?
Part 2: Assume that you are in the middle of running a *policy iteration* procedure over the Exercise 1 scenario, and that you are about to start a new *policy evaluation* step. If we were suddenly informed that we would have to use a new reward function from this point forward, would the policy iteration procedure ever converge to an optimal policy? Why?

Exercise 3 (bonus): Implementing Policy Iteration ||20/100 marks||

Implement the *policy iteration* algorithm and use it to find an optimal policy for the scenario in Exercise 1.

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

- the answers to any questions in the exercises in file `report.txt;`
- the script to run your solution for each exercise `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, `coursework1,`and then submit this directory using

```
submit rl cw1 coursework1
```

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.