Reinforcement Learning: Coursework 2018/19

Pavlos Andreadis, Arrasy Rahman, Stefano Albrecht

February 2019

Release date: Friday February 8th, 2019
Due date: 16:00 Friday March 29th, 2019

Introduction

This coursework is concerned with learning an optimal policy for the "Half-Field Offence" (HFO) problem (taken from `https://github.com/raharrasy/HFO`), with problem details defined in `https://github.com/raharrasy/RL2019-BaseCodes`. The latter also contains solution outlines and instructions on getting started for each exercise. Exercises 1, 2, and 4 ask you to solve a discretised version of the original problem, while exercise 3 exposes you to the original, continuous problem (you might find exercise 4 easier than 3 to tackle). Each exercise draws primarily from the following Reinforcement Learning (RL) topics:

- Exercise 1: Markov Decision Processes (MDPs), Dynamic Programming
- Exercise 2: Monte Carlo, Temporal-Difference Learning
- Exercise 3: Function Approximation, Deep RL
- Exercise 4: Multi-agent RL

Code & Report

The code for the exercises should be implemented in Python, and make use of the code available in the course repository `https://github.com/raharrasy/RL2019-BaseCodes`, and the course fork for the HFO repository `https://github.com/raharrasy/HFO`. Specifically, the exercises ask you to fill in certain scripts with supporting files defining the RL control problem that needs to be solved. The repository contains brief instructions on getting started with the code, while in-line comments indicate unimplemented functionality.

The submission does not involve a printed document or report. Where an implementation does not work correctly, comments on the code may

be taken into account positively. Submit any Python files you have written for the exercises, as well as a local version of the repository code, including any files you have modified. The solution for each exercise should be executable by running a specific script (as defined for each exercise), which you will fill in with your code. There is therefore no need to save and submit the results of running the scripts, with the exception of Exercise 3, which asks you to also submit training results (details on how to store the models are given in the Exercise 3 ReadMe on the repository). The submission should maintain the repository folder structure, copying all files into a folder named "coursework" (which is then submitted on DICE; see instructions at the end of this document).

Note, that this is individual and *not* group coursework.

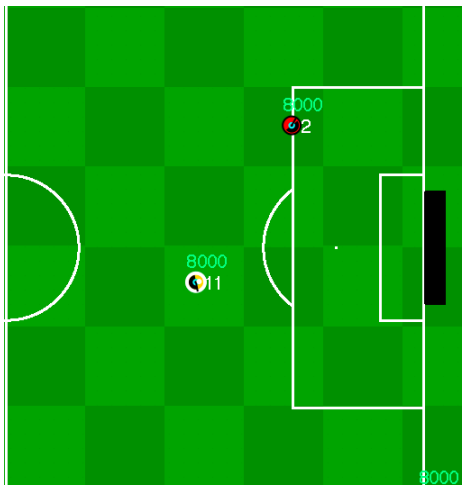
Evaluation

Your solutions will be evaluated on correctness (all exercises) as well as their performance level (exercises 2, 3, and 4). Performance is particularly emphasised for exercises 3, and 4. You are expected to use your understanding of the relevant algorithms and experimentation in order to select parameter values. Exercises are evaluated based on average return across episodes, with the exception of Exercise 3 that is evaluated on average time to goal (capped at the maximum number of time-steps per episode).

Exercise 1: Dynamic Programming ||20/100 marks||

Implement the *value iteration* algorithm for the discretised HFO problem as defined in the file `Exercise1\MDP.py`, by filling in the file `Exercise1\BellmanDPSolution.py`.

Exercise 2: Monte Carlo & TD Learning ||30/100 marks||



Solve the *control problem* for the discretised HFO problem, by filling in the files under the respective folder for each respective solution approach.

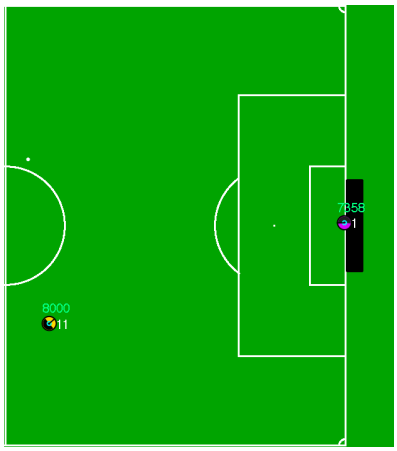
Part 1 (8 marks correct implementation, and 2 for performance): *Monte Carlo*,
`Exercise2\MonteCarlo\MonteCarloBase.py`

Part 2 (8 marks correct implementation, and 2 for performance): *SARSA*,
`Exercise2\SARSA\SARSABase.py`

Part 3 (8 marks correct implementation, and 2 for performance): *Q-Learning*,
`Exercise2\QLearning\QLearningBase.py`.

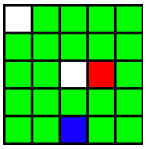
Exercise 3: Function Approximation & Deep RL ‖30/100 marks‖

(15 marks correct implementation, and 15 for performance)



Exercise 3 considers the problem of training a single-agent in the original, continuous HFO domain. Solve the *control problem* by implementing an agent which learns using an asynchronous Q-Learning approach. Pointers on the algorithm that should be implemented, metrics used to assess the performance of the agent, and list of functions that should be implemented are further provided in `Exercise3\README.md`.

Exercise 4: Multi-agent RL
‖20/100 marks‖



Exercise 4 considers a discretised soccer problem for 2 collaborative agents that have to be trained together. You are required to implement a team of agents trained with Independent Q-Learning (IQL), WoLF-PHC, and Joint Action Learning (JAL). Pointers on the algorithm that should be implemented, metrics used to assess the performance of the agent, and list of functions that should be implemented are further provided in `Exercise4\README.md`.

To solve this multi-agent *control problem* fill in the files under the respective folder for each respective solution approach.

Part 1 (4 marks correct implementation, and 2 for performance): *Independent Q-Learning*,
`Exercise4\IndependentQLearning\IndependentQLearning.py`

Part 2 (4 marks correct implementation, and 3 for performance): *WoLF-PHC*,
`Exercise4\WolfPHCAgent\WolfPHCBase.py`

Part 3 (4 marks correct implementation, and 3 for performance): *Joint Action Learning*,
`Exercise4\JointActionLearner\JointActionLearnerBase.py`.

Mechanics

Marks: This assignment will be assessed out of 100 marks and forms 25% of your final grade for the course.
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

- All the necessary changes to the base python codes as instructed in the `https://github.com/raharrasy/RL2019-BaseCodes`.

with names and folder structure of the provided structure left unchanged. You should copy all of the files to a single directory, `coursework`.Make sure

that the scripts are executable from this folder *as is*, without requiring of the marker to restructure your submission. Submit this directory using

```
submit rl cw1 coursework
```

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.

We strongly recommend that you plan to submit in advance of the deadline, as 5 mins late, will still count as a day late. Please do not contact the course staff with regard to the submit mechanism, or the system by which late penalties are applied.