RL 2021/22 Self-Check for Coding Proficiency

1 Purpose

The purpose of this document is to serve as a self-check to evaluate your coding proficiency and familiarise yourself with the expected coding skills required for the RL coursework. As part of the coursework, you will need to complete the following tasks:

- Install Python3 and standard ML libraries (e.g. Numpy and PyTorch)
- Create a virtual environment to organise dependencies (not required, but strongly suggested)
- Apply advanced knowledge of Python, including object-oriented programming (classes, constructors, functions), basic data structures (dictionaries, lists, Numpy arrays) and Python syntax

For latter exercises of the coursework implementing deep RL algorithms, you will be required to use PyTorch. We will provide a brief introduction to the basic knowledge needed, so no prior knowledge is required.

2 Coursework Overview

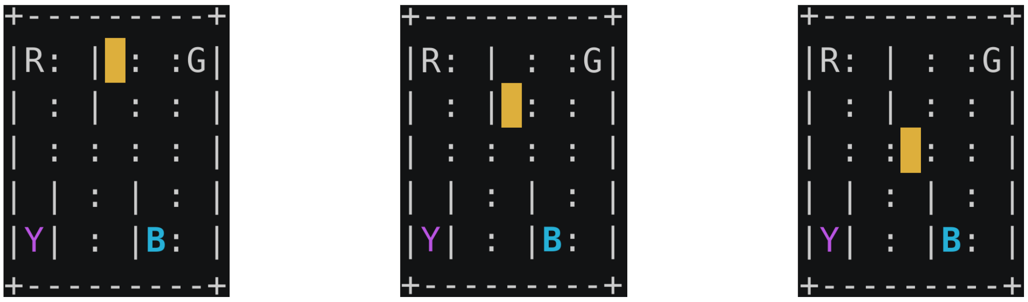
For the coursework, you will be required to provide implementations of RL algorithms within a predefined codebase that we provide. The codebase implements an interface for RL algorithms you will be tasked to implement specific functions in. You will need to familiarise yourself with the provided codebase, in particular understand its required interface (make sure you understand the expected in- and outputs of each function) and complete the implementation of specified functions. For some questions, besides implementing the required parts of an algorithm, we will also evaluate the performance of your learning agent in specified environments. For these performance evaluations, we will use your implementation of the RL algorithm with hyperparameters specified (and ideally optimised) by you and grade your submission using (1) correctness of the implementation of the algorithm and (2) performance marks where stated.

3 Example Question: Tabular-RL SARSA

This question serves as an example for a task in the RL coursework in which you will be asked to implement several RL algorithms. We suggest you go through this question carefully and evaluate whether you feel sufficiently confident in your coding skills to solve such a question (more context on the RL algorithms will be provided in the lectures).

SARSA For this question, you will need to implement the RL algorithm SARSA. You can find details on the algorithm in Section 6.4 of the RL book (pages 129-131).

Taxi-v3 We train the algorithm in the OpenAI Gym Taxi-v3 environment. In this environment, visualised in Figure 1, the goal is to navigate a taxi (yellow box - empty taxi; green box - taxi with passenger) to a passenger (blue location), pick it up and drop it off at the destination (purple location) in a grid-world. The task is successfully completed once the passenger is dropped off at its destination or failed after a maximum number of steps (which can be set as a hyperparameter). After each step, a numerical feedback in the form of a reward is provided. A reward of +20 is provided for successfully delivering the passenger to its destination, -10 for executing the actions pickup or dropoff illegally, i.e. trying to pickup a passenger at a location where no passenger is located or attempting to drop off without having a passenger in the taxi, and otherwise a reward of -1 is provided at each step. Hence, the task consists of learning to navigate the grid-world and bringing the passenger as quickly to its target destination as possible.



```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

Figure 1: Rendering of two Taxi-v3 environment steps

Implementation Use the codebase provided and implement the following functions. All of the functions that you need to implement are located in the `sarsa.py` file.

1. In the abstract `Agent` class, implement the following function:
   - `act`, where you must implement the ϵ-greedy exploration policy:

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

2. To implement SARSA, you must implement the following functions in the `SARSAAgent` class:
   - `learn`, where you must implement SARSA updates:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) + \alpha\left[r_{t+1} + \gamma Q\left(s_{t+1}, s_{t+1}\right) - Q\left(s_t, s_t\right)\right]$$

   - `schedule_hyperparameters`, where you can schedule the values of SARSA hyperparameters to improve performance.

All other functions apart from the aforementioned ones **should not be changed**.

Provided codebase You can find complementary code to this example question in the following code repository for the tutorial lecture. The provided code includes the general framework as well as an implemented solution for the SARSA RL algorithm with the following files:

- `sarsa.py`: Includes abstract `Agent` class and `SARSA` class. This is where the RL algorithm is implemented. You can find the respective solutions to required functions under comments stating `### SOLUTION BELOW ###`.
- `train_sarsa.py`: Includes creation of the environment and agent before starting the training procedure with frequent evaluations of the RL algorithm. This is the main script you execute to train and evaluate your agent. You will only need to adjust the hyperparameters within this script, found in the `CONFIG` dictionary.
- `requirements.txt`: List of required Python requirements needed to execute the code. These can be installed using `pip install -r requirements.txt`.

4 Demonstrations

As part of the RL course, there will be drop-in sessions where you will have the opportunity to ask questions regarding the coursework. Furthermore, there will be a lecture delivered by the TAs on the process of implementing and evaluating a RL system. A demonstration with more code will be provided as part of this lecture before the coursework is released. You can already find last years demonstration lecture resources under `https://github.com/uoe-agents/Building-a-Complete-RL-System_Demonstration` which will be updated for this year's tutorial lecture.