**Homework 2 - 15/11/17** **Deadline: 30/11/17 - 23:55**

CS 445/545

Instructor: Melih Kandemir
TA: Emre Göynügür (emre.goynugur@ozu.edu.tr)

## Homework 2: Mountain Car

In this assignment you are first asked to implement three algorithms (see below) using the Mountain Car (MountainCar.py) example code, which is uploaded to LMS. Please do not hesitate to contact the TA, if you have questions about the assignment.

- One-step semi-gradient Double Q-Learning (p. 147 (this is not the semi-gradient solution) - Sutton's Book - June 2017 Edition)
- N-step semi-gradient Sarsa (p. 259 - Sutton's Book - June 2017 Edition)
- N-step semi-gradient Expected Sarsa (p. 259 - Sutton's Book - June 2017 Edition)

Mountain Car is a problem in which an under-powered car must drive up a steep hill. Since gravity is stronger than the car's engine, even at full throttle, the car cannot simply accelerate up the steep slope. The car is situated in a valley and must learn to leverage potential energy by driving up the opposite hill before the car is able to make it to the goal at the top of the rightmost hill (taken from `https://en.0wikipedia.org/wiki/Mountain_car_problem`). In other words, the car has to increase by doing a swing motion (going up and down until reaches to the required speed).

After implementing the algorithms, you are asked to obtain and plot results for different values of n: 1 ... 8 (8 inclusive) for each algorithm you implemented. You can see how to evaluate your methods on page 260 (Figure 10.3) in Sutton's book (June 2017 Edition). This figure simply shows the number of steps the learning agent had to take to finish an episode. As it continues to learn, the number of steps required to complete an episode decreases. You could simply say that the better "performing" algorithm is the one that completes the game with the minimum number of steps.

Finally, you are asked to evaluate and present the performance of these algorithms in a one page report.

## Additional Notes

- The code *MCqlearn.py* on LMS implements DQN algorithm. Please **do not** try to change or submit this code.
- Rewards, state description and other functionalities required for your task are implemented in *MountainCar.py*. This file should provide all the necessary components to implement the three algorithms. However if you feel like you need to make changes, let us know.
  - An observation/state is a tuple that consists of a position and a velocity.
  - You can use a range [-1.2, 0.6] for the position variable. In this case, 0.5 could be your goal (right top of the hill).
  - You can use a range [-0.07, 0.07] for the velocity. If the velocity is negative, it means that the car is moving to left.

  - Your actions are 0: push left, 1: no push, 2: push right.
  - Reward is 100 if you reach the goal. Otherwise, -1 for all time steps.
  - An episode ends when you reach 0.5, or a certain amount of time steps is reached. i.e. 200 time steps.
  - You can start the game at a random point between -0.6 and -0.4 with no velocity.
  - You can use other configuration values. Given configuration values are taken from `https://github.com/openai/gym/wiki/MountainCar-v0`.

- Create a seperate file for each of the methods you implement. i.e. "double_q.py", "sarsa.py", "expected_sarsa.py". You can have other file(s) like "main.py", which implements the business logic of the assignment.
- If you haven't done plotting with Python before, you can follow tutorials here `https://matplotlib.org/users/beginner.html`.
- The GitHub page for the game can be found in `https://github.com/vikasjiitk/Deep-RL-Mountain-Car`.

## Code Snippets

```
# create an environment instance with the above setting
Xrange = [-1.2, 0.6]
Vrange = [-0.07, 0.07]
start = [random.uniform(-0.6, -0.4), 0]
goal = [0.5]

env = MountainCar(start, goal, Xrange, Vrange)
env.reset()

# get current state
curr_state = env.observe()

# i.e. apply an action: go to left
new_state, reward, game_over = env.act(0)
```

## Deliverables

- Your code that implements and benchmarks the three algorithms (**10 points**)
- A **one page** report of the comparative performance of the three RL models above and a mature interpretation of the outcome. (**5 points**)

## Late Submission Policy

- Less than 1 hour late: 1.5 points will be cut
- 1-24 hours late: 4 points cut
- 24-48 hours late: 8 points cut

- 48-72 hours late: 12 points cut
- Over 72 hours late: Submission not accepted

## Plagiarism and Submission Notes

- Team submissions are not allowed.
- Please do not exceed 1 page limit in your reports.
- Please do NOT copy from existing online solutions.
- There will be a careful review to detect plagiarism, especially in the RL algorithm. However, you may re-use functions (like game rules, drawing the board etc.).
- If you fail to pass the plagiarism test, you will fail the course with an F.