# HW10: Reinforcement Learning

## Extra Credit/Grading Note:

In this assignment we have an opportunity for extra credit. The assignment in Gradescope and Canvas will be out of 110 to reflect this. Your final assignments tab grade will be calculated as: min(assignments_score, 800). This means that you can make up for a maximum of 10 lost points in previous assignments using the extra credit from this assignment. **Note: It is possible to achieve a perfect score in the overall assignments section *without* doing the extra credit.**

## 1 Getting started

Download the starter code from Canvas. It consists of three main files: Q-Learning.py, tests.py, and visualize.py. For the extra credit part, there is an additional one file called SARSA.py. You can create and activate your virtual environment with the following commands:

```
python3 -m venv /path/to/new/virtual/environment

source /path/to/new/virtual/environment/bin/activate
```

Once you have sourced your environment, you can run the following commands to install the necessary dependencies:

```
pip install --upgrade pip

pip install gymnasium==0.29.1 matplotlib==3.9.2 pygame==2.5.2 numpy
```

You should now have a virtual environment which is fully compatible with the skeleton code. You should set up this virtual environment on an instructional machine to do your final testing.

## 2 Q-Learning on CliffWalking (100 points)

For the Q-learning portion of HW10, we will be using the environment CliffWalking-v0 from Farama Gymnasium. This is a discrete environment where the agent can move in the cardinal directions, and the agent must avoid falling over the cliff. The agent incurs a penalty of $-1$ each step and incurs a penalty of $-100$ if it falls into the cliff. Its goal is to reach the bottom right corner. You can read more about CliffWalking-v0 here: `https://gymnasium.farama.org/environments/toy_text/cliff_walking/`.

The main files for this part of the assignment are Q-learning.py, tests.py, and visualize.py. You will add your Q-learning code to Q-learning.py. **You will not need to change any code outside of the area marked TODO, but you are free to change the hyper-parameters if you want to.** For each sampled tuple $(s, a, r, s', \text{done})$, the update rule for Q-learning is:

$$Q(s,a) = \begin{cases} (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a')) & \text{if done == False} \\ (1-\alpha)Q(s,a) + \alpha r & \text{if done == True} \end{cases}$$

The agent should act according to an $\epsilon$-greedy policy as defined in the lecture notes. In this equation, $\alpha$ is the learning rate hyper-parameter, and $\gamma$ is the discount factor hyper-parameter. Once you've implemented the Q learning logic, the code should save the algorithm's Q table as Q_TABLE_QLearning.pkl.

- *HINT 1: tests.py is worth looking at to gain an understanding of how to use the Farama Gymnasium env.*
- *HINT 2: to visualize the learned policy based on the generated Q table, you can use visualize.py to see how the agent will behave in the environment given the generated Q table.* ***The visualization result will not be graded.***
- **Files to Submit:** For this section, you should submit the files Q_learning.py and Q_TABLE_QLearning.pkl.

# 3 Extra Credit: SARSA with Decaying Exploration (10 points)

The SARSA algorithm is very similar to Q-learning, although it differs in the fact that it is an *on-policy* algorithm. The name stands for "State, Action, Reward, State, Action" and refers to the tuples that are used by the update rule. For this extra credit portion, you will implement SARSA with decaying exploration on the CliffWalking-v0 environment.

You will add your code to the SARSA.py file. **You will not need to change any code outside of the area marked TODO, although, you are welcome to change hyperparameters you want.** For each sampled tuple $(s, a, r, s', a', \text{done})$, the update rule for SARSA is:

$$Q(s,a) = \begin{cases} (1-\alpha)Q(s,a) + \alpha(r + \gamma Q(s',a')) & \text{if done == False} \\ (1-\alpha)Q(s,a) + \alpha r & \text{if done == True} \end{cases}$$

Unlike the standard implementation, your SARSA agent must implement a decaying epsilon-greedy policy. Specifically:

1. Implement a schedule that decreases $\epsilon$ over time using the formula: $\epsilon = \epsilon_{\text{min}} + (\epsilon_{\text{max}} - \epsilon_{\text{min}}) \cdot e^{-\text{decay\_rate} \cdot \text{episode}}$
2. Set $\epsilon_{\text{min}} = 0.01$ and $\epsilon_{\text{max}} = 1.0$, and determine an appropriate decay rate.

This approach allows the agent to explore more extensively early in training and to increasingly exploit its learned policy as training progresses.

Once you've implemented the SARSA logic with decaying exploration, the code should save the algorithm's Q table as Q_TABLE_SARSA.pkl.

- **Files to Submit:** For this section, you should submit the files SARSA.py and Q_TABLE_SARSA.pkl.

# 4 Information

## 4.1 tests.py and visualize.py

tests.py and visualize.py are simply to help you with the assignment. They do not need to be changed and should not be submitted.

Once your Q_learning.py generates the Q-table, you can evaluate the performance of your RL agent by executing "python tests.py". It will tell you the average return of the agent. The file should automatically read your Q_TABLE_QLearning.pkl file. Similarly, visualize.py works the same way and can be executed by running "python visualize.py". visualize.py will visualize the policy that your agent follows based on the generated Q table.

In case you are working on the extra credit, you can apply tests.py and visualize.py by opening each file and changing the config field to ('CliffWalking-v0', 'SARSA').

### 4.2 Gymnasium Environment

You will need to use several Farama Gymnasium functions in order to operate your gym environment for reinforcement learning. As stated in a previous hint, tests.py has a lot of the function calls you need. Several important functions are as follows:

**env.step(action)**
Given that the environment is in state $s$, env.step takes an integer specifying the chosen action, and returns a tuple of the form $(s, r, \text{done}, \text{info})$. 'done' specifies whether or not $s'$ is the final state for that particular episode, and 'info' is unused in this assignment.

- Note: In the latest version of gym/gymnasium, 'done' is deprecated and replaced by more fine-grained return values that signal the end conditions of an epoch. But in this assignment with the environment you are in you should use 'done'.

**env.reset()**
Resets the environment to its initial state, and returns that state.

**env.action_space.sample()**
Samples an integer corresponding to a random choice of action in the environment's action space.

**env.action_space.n**
In the setting of the environments we will be working with for this assignment, this is an integer corresponding to the number of possible actions in the environment's action space.

You can read more about gymnasium here: `https://gymnasium.farama.org/api/env/`.

## 5 Submission

### 5.1 Due Date

**The assignment is due Wednesday April 30 at 11:59pm central time**. We are not accepting late submissions for this assignment.

### 5.2 Files

Please submit your file **Q_learning.py** and **Q_TABLE_QLearning.pkl** to Gradescope. You can test your learned policies, by calling python3 ./tests.py. Make sure to test your saved Q-tables using **tests.py** on the instructional machines with a virtual environment set up as specified above. This is the same program that we will be using to test your Q-tables and Q-network.

**Do not submit a Jupyter notebook .ipynb file, tests.py, visualize.py, nor any other extraneous files**.

Gradescope will test on various thresholds for score. You should aim for a score of $\geq -13$ on CliffWalking if you want to achieve full points.

## 5.3 Extra Credit

When submitting the extra credit part of the assignment, submit the additional files along with the main files submission: **SARSA.py** and **Q_TABLE_SARSA.pkl** to Gradescope.

## 5.4 Grading

Grading will be based on: 1) an actual implementation of the algorithms (Q learning); 2) how well the RL agent performs. For 2) we will award partial credit based on the average reward achieved by the agent for different thresholds, and full credit if the trained agent reaches the near optimal performance (e.g. $\geq -13$ in CliffWorld-v0).