COSC 6440 Deep Reinforcement Learning

HW1: Deep Q-Network (DQN)

Georgetown University, Spring 2025

Introduction

- In this homework, you will answer questions and implement the Q-Learning and Deep Q-Network (DQN) algorithm
- **Due date: Feb 21, 2025, 11:59 pm**
- Where to submit: **Canvas**
- What to submit: It is described at the end of this document.

Preparation

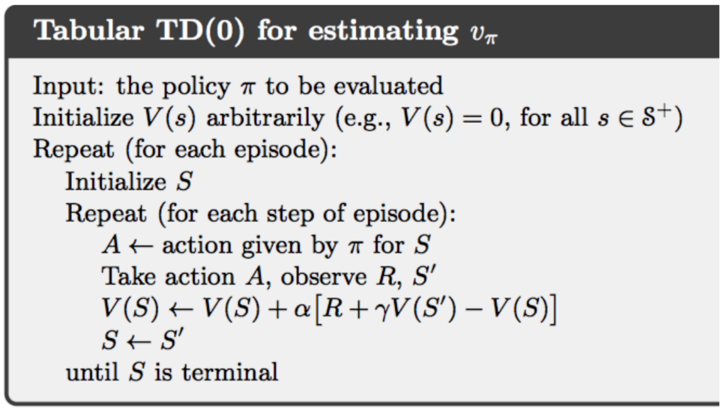
In RL, an agent interacts with an environment to accomplish a task. In this assignment, you will implement RL agents to run on robotic tasks.

- You will work in environments provided by the Gymnasium.
- To install the base Gymnasium library, use `pip install gymnasium`
- Use Python to finish this assignment.
- Your implementation may require Deep Learning (DL). Feel free to use any deep learning packages that you feel comfortable with. Both Tensorflow and Pytorch are acceptable.

Question 1.

Implement the following algorithms:

- TD(0)



```
Tabular TD(0) for estimating $v_\pi$

Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$, observe $R$, $S'$
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    until $S$ is terminal
```

- SARSA

```
**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
   Repeat (for each step of episode):
      Take action $A$, observe $R$, $S'$
      Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
      $S \leftarrow S'$; $A \leftarrow A'$;
   until $S$ is terminal
```

- Q-Learning

```
**Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$**

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Repeat (for each step of episode):
      Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      Take action $A$, observe $R$, $S'$
      $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
      $S \leftarrow S'$
   until $S$ is terminal
```

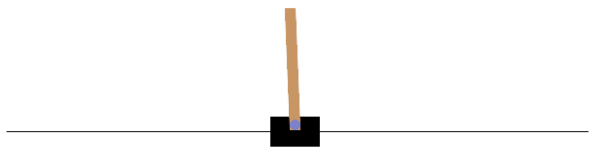
- DQN

```
**Algorithm 1: deep Q-learning with experience replay.**
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
**For** episode $= 1, M$ **do**
   Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
   **For** $t = 1$,T **do**
      With probability $\varepsilon$ select a random action $a_t$
      otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
      Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
      Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
      Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
      Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
      Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$
      Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the
      network parameters $\theta$
      Every $C$ steps reset $\hat{Q} = Q$
   **End For**
**End For**
```

For all the above algorithms, you can implement the ε-greedy algorithm for behavior policy to obtain the action action:

- With probability (1 - ε), take the best action indicated by the value functions or q-functions;
- With probability ε, take a random action;
- You can use ε=0.1.

Work on the following environment:



- **Environment**: CartPole-v1 (Click the link to find out more details)
- **Description**: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart.
- **Rewards**: Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted. The threshold for rewards is 500.

Plot the learning curves of your algorithms:

- Draw one learning curve for each algorithm and plot all learning curves on a single figure. Use proper legends and patterns of curves to distinguish each algorithm's curve.
- In the figure, the **x-axis should be the number of episodes**, and the **y-axis should be the undiscounted return** (i.e., the undiscounted sum of the reward of an episode) that your algorithm gains while it is learning
- Your figure should report at least **750 episodes**.
- You can choose to use a rolling average of the data to provide a smoother graph.

Question 2.

Write down what you have learned with your experimental results from Question 1. Analyze the performance of different algorithms in Question 1 by comparing the similarities and differences of these algorithms. Demonstrate your understanding and your insights.

<u>What to submit:</u>

Please **compress** the following into **a single .zip file and submit it to Canvas**.

- **Report.pdf** that contains your figure for Question 1 and your answer for Question 2
- **All your code** for this assignment.
- **Readme.txt** that contains the instructions to run your code, and libraries and dependencies to run your code;
- Any additional files that keep your code running normally.