Georgia Institute of Technology
Faculty of Interactive Computing

CS8803– ASSIGNMENT #1
Deep Reinforcement Learning

Instructor: Prof. Animesh Garg

Due date: 2024/10/22

Name: Ankit Sinha ______________________________
Student Number: 904050171 ______________________

This writing assignment contains 19 pages (including this cover page) and 28 questions. Total of points is 131.
Good luck and Happy reading work!

The assignment must be completed before **11:59 pm on Tuesday, Oct 22, 2024**. Upload your submission as one ZIP file on Canvas.

This assignment consists of two parts: a writing question and a coding question. For the writing question, your submission should be typeset, and all answers must be clearly legible. You are required to submit a PDF file containing your responses to the writing question (you may directly type your answer here and submit the file). For the coding question, you need to submit your code along with the running results, including the reward plot and a testing GIF file. Clearly specify which algorithm you are using to generate these plots. The coding question is divided into two sections: the fundamental coding questions (`hw1_basic.ipynb`) and the advanced coding questions (`hw1_advanced.ipynb`). In general, it is recommended that you complete the writing questions first, followed by the basic coding questions, and finally the advanced coding questions. Note that prior knowledge may be required to complete the advanced section. **Please ensure that you start this assignment as early as possible, as it will take time to run and fine-tune your code.**

The assignment must be completed by each student alone. Collaboration with other students is strictly prohibited. **Questions may be asked on Ed in case of unclear formulations. Do not post answers, partial answers or hints to answers!** Furthermore, we expect all students to adhere to the standards of academic integrity of the Georgia Institute of Technology.

1 Reinforcement Learning Fundamentals (10 points)

1. (3 points) Describe the difference between *supervised learning* and *reinforcement learning*. Use two application examples to highlight the difference.

   Ans: Supervised learning (SL) is a type of machine learning where the model is trained on labeled data, meaning each training example includes input-output pairs. The goal is to learn a mapping from inputs to outputs. In contrast, reinforcement learning (RL) involves an agent interacting with an environment to maximize a reward signal. In RL, the agent learns by trial and error rather than from a labeled dataset.

   1. SL in image classification. The model is trained on labeled images to classify them into categories.
   2. RL in robotics. A robot learns to walk by experimenting with different movements to maximize its reward (e.g., distance traveled without falling).

2. (4 points) Describe how one of your examples from the first question can be formalized with a MDP. State all formal parts of the MDP and sketch how the formal structure relates to your problem.

   Ans: To formalize the RL example of a robot learning to walk as an MDP, define the MDP components as follows:

   - State space S: the robot's possible states, such as its position and posture.
   - Action space A: the set of possible motor commands the robot can issue to move its legs.
   - Transition function $T(s, a, s')$: the probability of transitioning from state s to s' after taking action a.
   - Reward function $R(s, a)$: the reward the robot receives after taking action a in state s. For example, it may be proportional to the forward distance traveled.
   - Discount factor γ: used to prioritize immediate rewards over future rewards.

   The robot's goal in the MDP is to learn a policy $\pi(a|s)$ that maximizes the expected sum of discounted rewards over time.

3. (3 points) When dealing with *infinite horizon problems*, it is common to include a *discount factor*. Describe, in your own words, the role the discount factor plays in the reinforcement learning problem and what problem might occur if the discount factor is set to 1, and what problem might occur if the discount factor is set to 0.

   Ans: The discount factor $\gamma \in [0, 1]$ in RL controls the importance of future rewards relative to immediate rewards. A value of $\gamma = 1$ indicates the agent values future rewards equally as immediate rewards, which can lead to convergence issues as the sum of future rewards may become infinite in an infinite-horizon setting. If $\gamma = 0$, the agent ignores future rewards and focuses solely on immediate rewards, which may result in myopic behavior where long-term benefits are neglected.

2 Multiple Choices Questions (16 points)

For each of the following multiple-choice questions, select the correct answer(s). There may be more than one correct option per question. To simplify grading, the multiple-choice section will be uploaded to Canvas. **Complete this part on Canvas, as answers not submitted there will not be graded! We will open this section on Canvas as soon as possible!**

1. (2 points) Suppose we are having a well-defined finite grid world MDP problem with terminal state included. For each state, you will have your reward R defined properly and $R \geq 0$ for every grid. Suppose that we want to re-design the reward function there. For which of the following new reward functions would **guarantee** the optimal policy **remain unchanged**? Let $R(s, a, s')$ be the original reward function.

   ◯ $R_1(s, a, s') = 10R(s, a, s')$

   ◯ $R_2(s, a, s') = 1 + R(s, a, s')$

   ◯ $R_3(s, a, s') = R(s, a, s')^2$

   ◯ $R_4(s, a, s') = 1$

   ◯ None

2. (2 points) In TD learning with linear VFA (value function approximation), which of the following statement is true?

   ◯ To present a value function with a weighted linear combination of features, we can write $V(s; w) = x(s)^T w(s)$

   ◯ For SARSA using VFA, $\Delta w = \alpha(r + \gamma \hat{Q}(s', a'; w) - \hat{Q}(s, a; w))\nabla_w \hat{Q}(s', a'; w)$

   ◯ Consider we are using non-linear value function approximation, then during the weights update, we just need to re-calculate $\nabla_w \hat{Q}(s, a; w)$.

   ◯ None of the above

3. (2 points) Which statement is true about Policy gradient?

   ◯ Policy gradient can be computed with automatic differentiation when policy is a neural network. True

   ◯ Policy gradient is on-policy. True

   ◯ Policy gradient can also derive off-policy variants using importance sampling. True

   ◯ In Vanilla policy gradient, the condition number is small when computing the gradient. False

   ◯ In covariant/natual policy gradient, we re-scale the gradient properly to improve the optimization. True

4. (2 points) Which statement is true about Policy gradient? (Hint: two options are correct.)

   ◯ Policy gradient will have a low variance since gradient is unbiased.

- ◯ Policy gradient will have a high variance since huge amounts of rollouts will introduce noisy gradients.
- ◯ Using a larger batch in Policy gradient will lead to high variance.
- ◯ Tweaking the learning rate such as using adaptive step size rules can reduce the variance in Policy gradient.
- ◯ Having a fixed learning rate can reduce the variance in Policy gradient.

5. (2 points) Which statement is true about Policy gradient?
   - ◯ In Policy gradient, we subtract the baseline to reduce the variance. True
   - ◯ In Policy gradient, subtracting the baseline may introduce bias. False
   - ◯ In policy gradient, The baseline is essentially a proxy for the expected actual return. True
   - ◯ None of the above
6. (2 points) Which statement is true about Actor-critic?
   - ◯ Actor-Critic is a Temporal Difference(TD) version of Policy gradient.
   - ◯ Both actor and critic are improving overtime.
   - ◯ Consider actor-critic with discount, we prefer to discount the whole gradient because we care about the whole trajectory. False
   - ◯ We usually design actor and critic with shared features when states are high dimensional such as image state representation. True
   - ◯ In online actor-critic, we have both synchronized and asynchronous version.
7. (2 points) What statements are true about **on-policy** algorithms?
   - ◯ They are able to reuse data from different policies in the past directly
   - ◯ They are **not** able to reuse data from different policies in the past directly
   - ◯ They are always more sample efficient than off-policy algorithms
   - ◯ They cannot be used in discrete state-action spaces
   - ◯ None of the above
8. (2 points) Why do we need the log ratio trick in REINFORCE?
   - ◯ Because we cannot differentiate the gradient estimator otherwise
   - ◯ Because the original authors did not not like automatic differentiation
   - ◯ Because the gradient estimate has too much stochasticity
   - ◯ Because it helps with exploration
   - ◯ None of the above

3 Calculation Questions on Tabular Q-Leaning(8 points)

1. (4 points) Consider a system with two states and two actions. You perform actions and observe the rewards and transitions listed below. Each step lists the current state, reward, action and resulting transition as $S_i; R = r; a_k : S_i \to S_j$. Perform Q-learning using a learning rate $\alpha = 0.5$ and discount factor of $\gamma = 0.5$ for each step. The Q entries are initialized to zero. After step 3, what is $Q(S_2, a_1)$?

   Step 1: $S_1; R = -10; a_1 : S_1 \to S_1$
   Step 2: $S_1; R = -10; a_2 : S_1 \to S_2$
   Step 3: $S_2; R = +20; a_1 : S_2 \to S_1$

Given parameters:

$$\alpha = 0.5,$$
$$\gamma = 0.5.$$

The initial Q-values are set to zero:

$$Q(S_1, a_1) = 0,$$
$$Q(S_1, a_2) = 0,$$
$$Q(S_2, a_1) = 0,$$
$$Q(S_2, a_2) = 0.$$

Step-by-Step Calculation:

Step 1: $S_1; R = -10; a_1 : S_1 \to S_1$

$$\begin{aligned} Q(S_1, a_1) &\leftarrow Q(S_1, a_1) + \alpha \left(R + \gamma \max_a Q(S_1, a) - Q(S_1, a_1) \right) \\ &= 0 + 0.5\,(-10 + 0.5 \times 0 - 0) \\ &= -5. \end{aligned}$$

Step 2: $S_1; R = -10; a_2 : S_1 \to S_2$

$$\begin{aligned} Q(S_1, a_2) &\leftarrow Q(S_1, a_2) + \alpha \left(R + \gamma \max_a Q(S_2, a) - Q(S_1, a_2) \right) \\ &= 0 + 0.5\,(-10 + 0.5 \times 0 - 0) \\ &= -5. \end{aligned}$$

Step 3: $S_2; R = +20; a_1 : S_2 \to S_1$

$$
\begin{aligned}
Q(S_2, a_1) \leftarrow Q(S_2, a_1) + \alpha \left(R + \gamma \max_a Q(S_1, a) - Q(S_2, a_1)\right) \\
&= 0 + 0.5\,(20 + 0.5 \times \max(-5, -5) - 0) \\
&= 0.5 \times (20 - 2.5) \\
&= 0.5 \times 17.5 \\
&= 8.75.
\end{aligned}
$$

Final Answer:

$$Q(S_2, a_1) = 8.75.$$

2. (4 points) In the same setup as the previous question, what if you are performing SARSA algorithm and you are having an extra step 4 as followed. What is $Q(S_2, a_1)$ after performing the 4 steps?

   Consider a system with two states and two actions. You perform actions and observe the rewards and transitions listed below. Each step lists the current state, reward, action and resulting transition as $S_i; R = r; a_k : S_i \to S_j$. Perform SARSA using a learning rate $\alpha = 0.5$ and discount factor of $\gamma = 1$ for each step. The Q entries are initialized to zero.

   Step 1: $S_1; R = -10; a_1 : S_1 \to S_1$
   Step 2: $S_1; R = -10; a_2 : S_1 \to S_2$
   Step 3: $S_2; R = +20; a_1 : S_2 \to S_1$
   Step 4: $S_1; R = +10; a_1 : S_1 \to S_2$

Given parameters:

$$
\begin{aligned}
\alpha &= 0.5, \\
\gamma &= 1.
\end{aligned}
$$

The initial Q-values are set to zero:

$$
\begin{aligned}
Q(S_1, a_1) &= 0, \\
Q(S_1, a_2) &= 0, \\
Q(S_2, a_1) &= 0, \\
Q(S_2, a_2) &= 0.
\end{aligned}
$$

Step-by-Step Calculation:

Step 1: $S_1; R = -10; a_1 : S_1 \to S_1$

$$\begin{aligned} Q(S_1, a_1) &\leftarrow Q(S_1, a_1) + \alpha \left(R + \gamma Q(S_1, a_1) - Q(S_1, a_1)\right) \\ &= 0 + 0.5\left(-10 + 1 \times 0 - 0\right) \\ &= -5. \end{aligned}$$

Step 2: $S_1; R = -10; a_2 : S_1 \to S_2$

$$\begin{aligned} Q(S_1, a_2) &\leftarrow Q(S_1, a_2) + \alpha \left(R + \gamma Q(S_2, a_1) - Q(S_1, a_2)\right) \\ &= 0 + 0.5\left(-10 + 1 \times 0 - 0\right) \\ &= -5. \end{aligned}$$

Step 3: $S_2; R = +20; a_1 : S_2 \to S_1$

$$\begin{aligned} Q(S_2, a_1) &\leftarrow Q(S_2, a_1) + \alpha \left(R + \gamma Q(S_1, a_1) - Q(S_2, a_1)\right) \\ &= 0 + 0.5\left(20 + 1 \times (-5) - 0\right) \\ &= 0.5 \times (20 - 5) \\ &= 0.5 \times 15 \\ &= 7.5. \end{aligned}$$

Step 4: $S_1; R = +10; a_1 : S_1 \to S_2$

$$\begin{aligned} Q(S_1, a_1) &\leftarrow Q(S_1, a_1) + \alpha \left(R + \gamma Q(S_2, a_1) - Q(S_1, a_1)\right) \\ &= -5 + 0.5\left(10 + 1 \times 7.5 + 5\right) \\ &= -5 + 0.5\left(10 + 7.5\right) \\ &= -5 + 0.5 \times 17.5 \\ &= -5 + 8.75 \\ &= 3.75. \end{aligned}$$

Final Answer:

$$Q(S_2, a_1) = 7.5.$$

4 Analyzing a Modern Algorithm: DDPG (14 points)

In this section, we will work our way towards the formulation of a modern high performance reinforcement learning algorithm. As we shall see, this algorithm is ultimately derived from similar principles as the reinforcement learning algorithms discussed in class, with adjustments to increase its stability and efficiency. In particular, we will discuss the Deep Deterministic Policy Gradient (DDPG) algorithm. The DDPG algorithm can be found in a simplified form in Figure 1.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$

repeat

  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$

  Execute a in the environment

  Observe next state s', reward r, and done signal d to indicate whether s' is terminal

  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$

  If s' is terminal, reset environment state.

  if it's time to update **then**

    for however many updates **do**

      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for

  end if

until convergence

Figure 1: DDPG Algorithm

1. (3 points) State a similarity/relationship of this algorithm to **2 different algorithms** that were discussed in class, **one similarity per algorithm**.

Ans: The DDPG algorithm shares similarities with both Q-learning and Actor-Critic methods:

1. **Similarity with Q-learning:** DDPG uses an off-policy approach similar to Q-learning, where it stores experience tuples in a replay buffer and samples them for training. This off-policy nature allows for stable training as the target values are computed independently of the current policy.

2. **Similarity with Actor-Critic:** DDPG is based on the Actor-Critic architecture, where it combines both actor and critic networks. The actor network determines the policy by selecting continuous actions, and the critic network estimates the Q-values, which guide the actor's policy improvement.

2. (4 points) DDPG aims to learn a Q-function within a continuous action space. Of course, we learned in class that this is generally quite difficult.
(i) (1 point) Please state the reason that this is so.
(ii) (3 points) Subsequently, state and explain the approach adopted by DDPG to avoid the issue you raised, with reference to the algorithm written above.

1. **Reason for Difficulty:** Learning a Q-function in a continuous action space is challenging because finding the optimal action a that maximizes the Q-value, $Q(s, a)$, requires an exhaustive search over all possible actions. In continuous spaces, this maximization is computationally infeasible as there are infinitely many possible actions.

2. **Approach adopted by DDPG:** To address this issue, DDPG uses an actor-critic architecture where the actor network is responsible for selecting actions directly, rather than performing an exhaustive maximization over Q-values. The critic network estimates the Q-value $Q(s, a)$ for the actions proposed by the actor. This allows DDPG to operate efficiently in continuous action spaces without requiring exhaustive searches. Furthermore, DDPG uses target networks and experience replay to stabilize the training process by reducing correlations in the updates.

3. (5 points) State and explain one alternate approach to solve the issue raised in the previous part of this question, using pseudo-code to illustrate your idea. What is one benefit and one drawback of this approach compared to DDPG?

Alternate Approach: Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an alternative approach for learning in continuous action spaces. SAC incorporates entropy regularization to encourage exploration by maximizing both the expected reward and the entropy of the policy. This helps prevent premature convergence to suboptimal policies.

Pseudo-Code for SAC:

```
for each update step do
    Sample a batch of transitions (s, a, r, s') from the replay buffer
    Calculate target value: y = r +  * (Q(s', (s')) -  * log((a'|s')))
    Update the Q-function by minimizing: (y - Q(s, a))^2
    Update the policy by maximizing: Q(s, (s)) +  * entropy((s))
    Adjust temperature parameter  to control entropy
end for
```

Benefit: SAC's entropy regularization allows for more robust exploration, which often leads to better policies in stochastic environments or tasks with sparse rewards.

Drawback: The addition of entropy regularization increases the computational complexity and may slow down convergence in environments where exploration is less critical.

4. (2 points) The policy portion of the update is not the same as the standard policy gradient, and instead makes an assumption on some property of the Q function.

   State the assumption on Q that DDPG is using here for their policy update, and explain how it justifies the expression in gradient ascent.

Assumption on Q in DDPG:

DDPG assumes that the Q-function $Q(s, a)$ is differentiable with respect to the action a. This allows the policy gradient to be computed by directly maximizing the Q-value with respect to the action, i.e., using the gradient $\nabla_a Q(s, a)$.

Justification:

Under this assumption, the policy update can be performed by taking the gradient of the Q-function with respect to the actions selected by the actor. Specifically, the actor parameters θ are updated by ascending the gradient of $Q(s, \pi(s; \theta))$, which directs the actor towards actions that maximize the expected Q-value. This enables efficient policy improvement by directly leveraging the Q-function to guide action selection without relying on sampling-based policy gradients.

5 Reinforcement Learning Application (18 points)

Reinforcement learning method can be widely used in many different field. For this question, we will explore deep reinforcement learning methods in traffic signal control (TSC) applications. Consider a simple TSC problem. In a single four-way intersection, each direction is controlled with green, red and yellow phases by a traffic light. The main goal of TSC is providing safe, effective and reliable transportation systems to participants.

1. (2 points) Please briefly state that why TSC problem can be described as MDP.

   Answer:

   The Traffic Signal Control (TSC) problem can be described as a Markov Decision Process (MDP) because it consists of states, actions, transitions, and rewards, which are core components of an MDP. In TSC:

   - **State:** Represents the current traffic conditions (e.g., vehicle positions, waiting times) at the intersection.
   - **Action:** Corresponds to the traffic light configurations (e.g., green or red for each direction).
   - **Transition:** Defines how the state changes in response to an action (e.g., changing the light can alter vehicle flow).
   - **Reward:** Quantifies the effectiveness of actions (e.g., reducing waiting times or improving traffic flow).

   By modeling the TSC problem as an MDP, reinforcement learning algorithms can be used to learn optimal policies for controlling traffic lights.

2. (4 points) Dealing with the TSC problem on simulators by using RL algorithms requires a good problem formulation in several parts: state, action, reward definitions and neural network structure. In this single four-way intersection situation, please describe what is your state and action in this problem and give a concrete example of each. You need to describe your state and action clearly on what each feature in your state and action represent of and how you are extracting this information.

   Answer:

   State:

   The state in the TSC problem represents information about the traffic conditions at a four-way intersection. For example, the state vector could include:

   - The number of vehicles waiting in each lane.
   - The average waiting time of vehicles in each lane.
   - The current traffic light configuration (e.g., which lanes are currently green).

A concrete example of the state could be:

$$s = [5, 3, 10, 2, 1, 0, 0, 1]$$

where each entry represents, respectively, the number of vehicles waiting in each lane and the current traffic light configuration (binary values indicating green or red for each direction).

Action:

The action in this TSC problem specifies the traffic light configuration for the intersection. For instance, actions could be defined as:

- Switching the lights to allow traffic flow from the north-south direction.
- Switching the lights to allow traffic flow from the east-west direction.
- Maintaining the current light configuration for a fixed duration.

A concrete example of an action might be:

$$a = \text{"NS green, EW red"}$$

where "NS" represents north-south lanes and "EW" represents east-west lanes.

3. (4 points) The role of reward in RL is analyzing the quality of taken action with respect to the state. Please list at least 2 possible scenarios that might happen in this TSC problem and write the mathematical representation (such as $r_0 = \sum_i A_i^2$ where A is ...) to describe how you are going to penalize or award the agent for this scenario.

Answer:

Scenario 1: Reducing Vehicle Waiting Time

In this scenario, the reward r can be based on minimizing the total waiting time of all vehicles at the intersection:

$$r = -\sum_{i=1}^{N} w_i$$

where w_i is the waiting time for the i-th vehicle, and N is the total number of vehicles at the intersection. A lower total waiting time will yield a higher (less negative) reward, encouraging the agent to reduce delays.

Scenario 2: Avoiding Congestion

Another scenario could involve penalizing the agent for high vehicle density in any lane, indicating potential congestion. The reward could be:

$$r = -\sum_{j=1}^{M} d_j^2$$

where d_j represents the density of vehicles in lane j, and M is the number of lanes. By penalizing the square of the vehicle density, this reward structure discourages congestion and promotes even traffic flow across lanes.

4. (3 points) Noticing that a car accident must be prevented in TSC. We add a safety constraint $\mathcal{C} = \{c : S \to \{0, 1\}\}$ that the agent must satisfy. With this extra constraint, we call it constrained MDP (CMPD). One could also argue that we can add a penalty to the agent when car accident happens. Please clarify what is the difference between C-MDP and adding reward penalty.

Answer: In a Constrained MDP (CMDP), the agent is explicitly required to satisfy the safety constraint C, which prevents the occurrence of unsafe states (such as car accidents). This means that the agent's policy must avoid any actions that could lead to accidents, enforcing a strict rule.

On the other hand, adding a reward penalty for accidents is a softer approach. Here, the agent is penalized when an accident occurs, but there is no guarantee that the agent will completely avoid unsafe actions. The penalty discourages but does not prohibit the agent from taking risky actions that may lead to accidents.

In summary:

- CMDP enforces a hard constraint on safety.

- Adding a reward penalty discourages, but does not strictly prevent, unsafe actions.

5. (3 points) Supposed that we are using Policy Optimization method to solve the CMDP problem describing above. What could be a problem compared to MDP?

Answer: In a CMDP, the agent has to satisfy additional constraints on the policy, such as avoiding unsafe actions (e.g., preventing car accidents). This introduces a trade-off between maximizing rewards and satisfying constraints, which can complicate the optimization process. Specifically, the agent may need to sacrifice some reward to comply with the safety constraint, leading to a more complex learning problem than in a standard MDP, where the agent only needs to maximize rewards.

Additionally, policy optimization methods often rely on gradient-based approaches, which may struggle with balancing the primary reward objective with strict constraints, leading to slower convergence or even infeasible solutions if the constraints are not easily satisfied.

6. (2 points) Please briefly describe one intuition to solve the problem in the previous question.

Answer: One intuition to address the constraint challenge in CMDPs is to use a Lagrangian relaxation approach, where the constraint is incorporated into the objective function by adding a penalty term. Specifically, the agent maximizes a modified objective:

$$\mathcal{L}(\pi) = \mathbb{E}[R(s,a)] - \lambda\mathbb{E}[C(s,a)]$$

where λ is a Lagrange multiplier that balances the reward and constraint terms. The value of λ can be adjusted during training to ensure the agent respects the safety constraint while still aiming to maximize the reward, thereby transforming the constrained problem into an unconstrained one.

6 Coding Questions (65 points)

This section provides instructions for the coding assignment. The coding questions are divided into two parts: basic coding questions (covering algorithms taught in class) and advanced coding questions (implementing a commonly used model-free algorithm, not covered in class but guided step-by-step). Please clone the repository from `https://github.com/pairlab/cs8803drl-fall24.git` to access the code. Follow the instructions provided to complete the coding assignment step by step.

Before beginning the coding tasks, follow the instructions in the `README.md` file to set up a Python environment for running the code. You may use either a GPU or CPU to train your algorithm.

Notice: All hyperparameters inside the `Hyper-parameters Tuning` cell may require adjustment. Please allocate sufficient time to complete the coding questions.

You are free to add any additional functions or methods to the code if you find that simply completing the TODO sections is not sufficient to make the code run properly.

6.1 Basic Coding Questions (40 points)

For the basic coding question, the code is provided in `hw1_basic.ipynb`. Ensure that you run all cells in this notebook and save the reward plot and GIF file as `{algo_name}_returns.png` and `{algo_name}_policy.gif`, respectively. Follow the instructions for each algorithm to complete the corresponding functions in the `/src` directory.

If you run the code cells in `hw1_basic.ipynb`, `{algo_name}_returns.png` and `{algo_name}_policy.gif` will be automatically generated under the `artifacts/` folder. In the basic coding task, we are using the "LunarLander-v2" environment with discrete action space. You are encouraged to try different discrete or continuous action environments in Gym. You can also explore more complex tasks from Gym-Atari or MuJoCo (note that these may require additional package installations).

1. (10 points) **REINFORCE algorithm:**
   Completing the code in `src/network.py` and `src/reinforce.py` based on the TODO instructions should be sufficient to run the REINFORCE algorithm. You can tune the parameters inside the `Hyper-parameters Tuning` cell in the `hw1_basic.ipynb` file if you find that the provided values are not sufficient to achieve the desired results.

2. (10 points) **Simple Q-iteration (no experience replay + target network):**
   Completing the code in `src/network.py` and `src/q_iter.py` based on the TODO instructions should be sufficient to run the Q-iteration algorithm. You can tune the parameters inside the `Hyper-parameters Tuning` cell in the `hw1_basic.ipynb` file if you find that the provided values are not sufficient to achieve the desired results.

3. (10 points) **DQN (Deep Q-learning + experience replay + target network):**
   Completing the code in `src/network.py` and `src/dqn.py` based on the TODO instructions should be sufficient to run the DQN algorithm. You can tune the parameters inside the

`Hyper-parameters Tuning` cell in the `hw1_basic.ipynb` file if you find that the provided values are not sufficient to achieve the desired results.

4. (10 points) **Actor-Critic:**
   Completing the code in `src/network.py` and `src/ac.py` based on the TODO instructions should be sufficient to run the Actor-Critic algorithm. You can tune the parameters inside the `Hyper-parameters Tuning` cell in the `hw1_basic.ipynb` file if you find that the provided values are not sufficient to achieve the desired results.

6.2 Advanced Coding Questions (25 points)

For the advanced coding question, the code is provided in `hw1_advanced.ipynb`. Ensure that you complete and run all cells in this notebook and save the reward plot `sac_returns.png`.

In the advanced coding task, we are using the "Pendulum-v1" environment with continuous action space. You are encouraged to try different continuous action environments in Gym. You can also explore more complex tasks from Gym-Atari or MuJoCo (note that these may require additional package installations).

1. (25 points) **Soft Actor-Critic:**
   Complete the code in `hw1_advanced.ipynb` based on the TODO instructions. You can tune the parameters inside the `Hyper-parameters Tuning` cell in the `hw1_advanced.ipynb` file if you find that the provided values are not sufficient to achieve the desired results.

Checkbox for Assignment Submission

Congratulations on completing Assignment 1! We have created a checklist for you to double-check your submission. The files you need to submit are:

- ☐ **hw1.zip**: a zip file with your completed PDF and code.
  - ☐ **Python files**:
    `ac.py`, `dqn.py`, `networks.py`, `q_iter.py`, `reinforce.py`
  - ☐ **Jupyter Notebook files**:
    `hw1_basic.ipynb`, `hw1_advanced.ipynb`
  - ☐ **Coding results**:
    `ac_policy.gif`, `ac_returns.png`,
    `dqn_policy.gif`, `dqn_returns.png`,
    `q_iteration_policy.gif`, `q_iteration_returns.png`,
    `reinforce_policy.gif`, `reinforce_returns.png`,
    `sac_returns.png`, `sac_policy.gif` (optional)
- ☐ **Writing assignment PDF file**: `CS8803_DRL_A1.pdf`
- ☐ **hw1_basic.pdf**:
  Convert the `hw1_basic.ipynb` to a PDF file **containing your running results.**
- ☐ **hw1_advanced.pdf**:
  Convert the `hw1_advanced.ipynb` to a PDF file **containing your running results.**
- ☐ **Completing the Multiple Choices Questions on Canvas.**