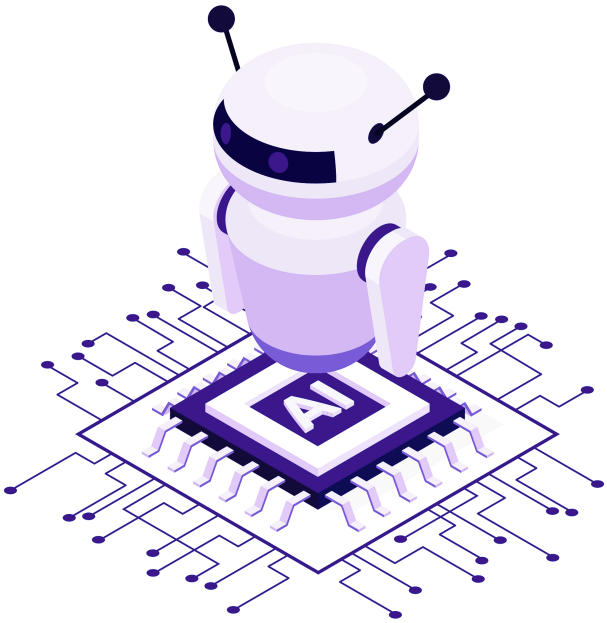

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

Designed By:

Reza GhaderiZadeh

r.ghaderi2001@gmail.com

Dariush Jamshidian

drjm313@gmail.com

RIML

Deep RL Course

Spring 2025

Preface

Welcome to this homework assignment on Deep Reinforcement Learning! In this set of tasks, you will delve into two powerful reinforcement learning techniques: Value-Based Methods, specifically Deep Q-Networks (DQN) and Double DQN (DDQN), as well as N-Step Sarsa and N-Step Q-Learning. Through these exercises, you will gain hands-on experience implementing and evaluating algorithms in challenging reinforcement learning environments.

The primary goal of this assignment is to provide you with a deeper understanding of how value-based methods can be used to approximate action-value functions and how modifications like Double DQN help in improving performance by reducing biases. You will also explore the impact of step-sizes in N-Step Sarsa and N-Step Q-learning to enhance your grasp of temporal difference methods in reinforcement learning.

To achieve this, you will be working with widely recognized benchmark environments like CartPole and Cliff Walking, which will help in testing and comparing the performance of the algorithms you implement.

By the end of this homework, you will have:

- Gained practical experience implementing DQN and DDQN to solve a control problem.
- Developed an understanding of the trade-offs between different reinforcement learning methods through the analysis of performance and stability.
- Explored the benefits and challenges associated with N-step methods and learned to fine-tune algorithms for optimal performance.

This homework will not only strengthen your coding and analytical skills but also give you an insight into the theory and practical implementation of reinforcement learning algorithms, which are foundational for more advanced topics in the field.

We encourage you to make the most of this exercise by experimenting with different configurations, hyperparameters, and environments. Remember that reinforcement learning often involves exploration and trial-and-error so don't hesitate to test various setups to better understand the nuances of these algorithms.

Good luck, and we look forward to seeing your solutions!

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: Epsilon Greedy & N-step Sarsa/Q-learning	40
Jupyter Notebook	25
Analysis and Deduction	15
Task 2: DQN vs. DDQN	50
Jupyter Notebook	30
Analysis and Deduction	20
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

* Note that the sum of the grades in the notebooks is out of 100 and will be scaled to the above scores.

Submission

The deadline for this homework is 1403/12/5 (February 23th 2025) at 11:59 PM.

Please submit your work by following the instructions below:

- Place your solution alongside the Jupyter notebook(s).
  - Your written solution must be a single PDF file named `HW2_Solution.pdf`.
  - If there is more than one Jupyter notebook, put them in a folder named `Notebooks`.
- Zip all the files together with the following naming format:

  `DRL_HW2_[StudentNumber]_[FullName].zip`

  - Replace `[FullName]` and `[StudentNumber]` with your full name and student number, respectively. Your `[FullName]` must be in CamelCase with no spaces.
- Submit the zip file through Quera in the appropriate section.
- We provided this LaTeX template for writing your homework solution. There is a 10-point bonus for writing your solution in LaTeX using this template and including your LaTeX source code in your submission, named `HW2_Solution.zip`.
- If you have any questions about this homework, please ask them in the Homework section of our Telegram Group.
- If you are using any references to write your answers, consulting anyone, or using AI, please mention them in the appropriate section. In general, you must adhere to all the rules mentioned here and here by registering for this course.

Keep up the great work and best of luck with your submission!

Contents

1 Epsilon Greedy & N-step Sarsa/Q-learning

This section of the homework involves working with two reinforcement learning algorithms, N-step Sarsa and N-step Q-learning, applied to the Cliff Walking environment. You will be required to complete a Jupyter notebook where these algorithms are implemented and tested. Afterward, you will analyze your observations and draw conclusions based on the results you obtain. This task will help deepen your understanding of these algorithms and their performance in a standard reinforcement learning environment.

1.1 Cliff Walking Environment

The primary environment used to test these algorithms will be the Cliff Walking environment from OpenAI's Gym. The environment consists of a grid where an agent must navigate from a starting point to a goal point while avoiding falling off a cliff. The agent receives a negative reward for falling into the cliff, making the task challenging. You can explore the environment and learn more about it through the following link: https://gymnasium.farama.org/environments/toy_text/cliff_walking/.

This environment provides a great testing ground for reinforcement learning algorithms because it requires the agent to balance exploration and exploitation over multiple steps.

1.2 Tasks and Objectives

There are two main tasks that you need to accomplish:

1.2.1 Task 1: Completing the Jupyter Notebook

You will be provided with a Jupyter notebook containing the initial framework for both N-step Sarsa and N-step Q-learning. Your task is to complete the notebook by implementing both algorithms and testing them on the Cliff Walking environment. Ensure to:

- Implement Epsilon Greedy algorithm.
- Implement the N-step Sarsa algorithm.
- Implement the N-step Q-learning algorithm.
- Experiment with different values of epsilons.
- Evaluate both algorithms on the Cliff Walking environment and track their performance.
- Experiment with different values of N to see how it affects the performance of each algorithm.

1.2.2 Task 2: Analysis and Deduction

Once you have implemented the algorithms and obtained results, you will need to analyze the outcomes. This task involves comparing the performance of N-step Sarsa and N-step Q-learning, and making a deduction based on the observations. You should:

Epsilon Greedy

- Analyze the regret plots:
  - Epsilon 0.1 initially has a high regret rate but decreases quickly. Why is that?
  - Both epsilon 0.1 and 0.5 show jumps. What is the reason for this?
  - Epsilon 0.9 almost changes linearly. Why?
- Compare the optimal policy for epsilon values 0.1 and 0.9. How do they differ, and why do they look different?
- In the epsilon decay section, analyze the optimal policy for the row adjacent to the cliff(lowest row). Then, compare the different learned policies and their corresponding rewards.

N-step Sarsa and N-step Q-learning

- What is the difference between Q-learning and sarsa?
- Compare how different values of n affect each algorithm's performance separately.
- Is a Higher or Lower n Always Better? Explain the advantages and disadvantages of both low and high n values.

2 DQN vs. DDQN

This homework section involves working with two reinforcement learning algorithms, Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN), applied to a standard RL environment. You will complete a Jupyter notebook where these algorithms are implemented and tested. Afterward, you will analyze the performance of each approach and draw conclusions based on your results. This exercise will help deepen your understanding of function approximation in reinforcement learning and the challenges related to Q-value estimation.

2.1 Overview of DQN and DDQN

Before starting the implementation, it is important to understand what DQN and DDQN are and how they differ. Both methods use deep neural networks to approximate Q-values but differ in how they address the problem of overestimation bias.

2.1.1 Deep Q-Networks (DQN)

DQN is a value-based reinforcement learning algorithm that utilizes a deep neural network to approximate the action-value function $Q(s, a)$. It employs experience replay and a target network to stabilize training. However, DQN is known to suffer from overestimation bias, as it selects and evaluates actions using the same Q-network, leading to unstable training in some environments.

2.1.2 Double Deep Q-Networks (DDQN)

DDQN is an improvement over DQN that aims to reduce the overestimation of Q-values. It achieves this by using one network to select actions (online network) and another to evaluate their values (target network), leading to more stable learning. This modification helps in environments where Q-value overestimation negatively impacts learning.

2.2 CartPole Environment

The CartPole environment from OpenAI Gymnasium will be used to evaluate the performance of DQN and DDQN. In this environment, an agent controls a cart moving on a track with a pole balanced on top. The goal is to apply left or right forces to keep the pole upright for as long as possible. The episode ends when the pole falls beyond a threshold angle or when the cart moves out of bounds. You can explore the environment and learn more about it through the following link:
https://gymnasium.farama.org/environments/classic_control/cart_pole/

2.3 Tasks and Objectives

There are two main tasks in this section:

2.3.1 Task 1: Completing the Jupyter Notebook

You will be provided with a Jupyter notebook containing a partially implemented framework for both DQN and DDQN. Your task is to complete the missing sections (marked as TODO) and ensure the algorithms function correctly. Specifically, you will:

- Implement the missing components of the DQN algorithm
- Implement the missing components of the DDQN algorithm.
- Train both models.

2.3.2 Task 2: Analysis and Evaluation

Once you have completed the implementations and trained the models, analyze their performance by considering the following aspects:

- Which algorithm performs better and why?
- Which algorithm has a tighter upper and lower bound for rewards.
- Based on your previous answer, can we conclude that this algorithm exhibits greater stability in learning? Explain your reasoning.
- What are the general issues with DQN?
- How can some of these issues be solved? (You may refer to external sources such as research papers and blog posts be sure to cite them properly.)
- Based on the plotted values in the notebook, can the main purpose of DDQN be observed in the results?
- The DDQN paper states that different environments influence the algorithm in various ways. Explain these characteristics (e.g., complexity, dynamics of the environment) and their impact on DDQNs performance. Then, compare them to the CartPole environment. Does CartPole exhibit these characteristics or not?
- How do you think DQN can be further improved? (This question is for your own analysis, but you may refer to external sources such as research papers and blog posts—be sure to cite them properly.)

3 Appendix: Pseudo Code for N-Step Sarsa and N-Step Off-policy Learning

Below are the pseudo codes for both N-step Sarsa and N-step off-policy learning based on Sutton's book.

3.1 Pseudo Code for N-Step Sarsa

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

3.2 Pseudo Code for N-Step Off-policy learning

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$, then:
      Take action A_t
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      If S_{t+1} is terminal, then:
        $T \leftarrow t+1$
      else:
        Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ ($\rho_{\tau+1:\tau+n}$)
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ ($G_{\tau:\tau+n}$)
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho[G - Q(S_\tau, A_\tau)]$
      If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

References

[1] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd Edition, 2020. Available: http://incompleteideas.net/book/the-book-2nd.html.

[2] Gymnasium Documentation. Available: https://gymnasium.farama.org/

[3] Grokking Deep Reinforcement Learning. Available: https://www.manning.com/books/grokking-deep-reinforcement-learning

[4] Deep Reinforcement Learning with Double Q-learning. Available: https://arxiv.org/abs/1509.06461

[5] Cover image designed by freepik