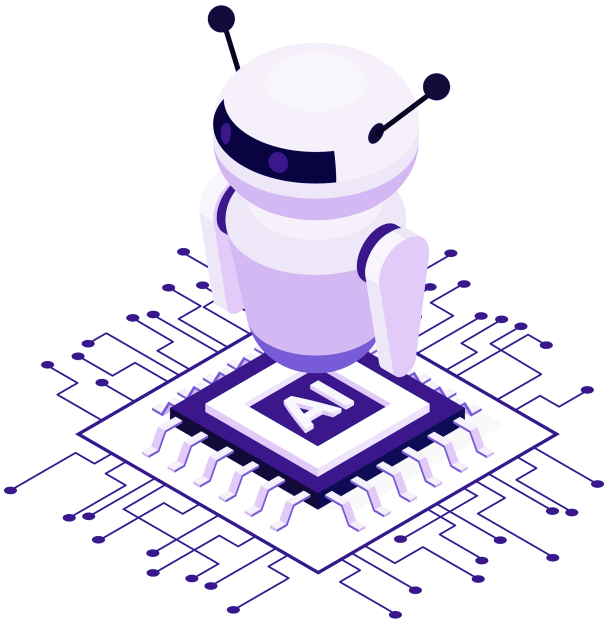

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 5:

Model-Based Reinforcement Learning

Designed By:

Naser Kazemi

naserkazemi2002@gmail.com

Ramtin Moslemi

ramtin4moslemi@gmail.com

RIML

Spring 2025

Preface

Welcome to this homework assignment on Deep Reinforcement Learning! In this set of tasks, you will explore three fundamental approaches to model-based and model-free reinforcement learning: **Monte Carlo Tree Search (MCTS)**, **Dyna-Q**, and **Model Predictive Control (MPC)**. Through these exercises, you will gain hands-on experience implementing and evaluating algorithms in various reinforcement learning environments.

The primary goal of this assignment is to provide you with a deeper understanding of how planning and search methods can enhance decision-making and how integrating learned models or policies can further improve performance. You will explore how **Monte Carlo Tree Search** balances exploration and exploitation, how **Dyna-Q** efficiently combines real and simulated experiences, and how **MPC** leverages predictive models for control in continuous-action settings.

To achieve this, you will be working with diverse scenarios — from discrete-action puzzles suitable for MCTS to continuous-control domains suited for MPC. By the end of this homework, you will have:

- **Gained practical experience implementing MCTS**, understanding its role in decision-making and strategic planning.
- **Developed an understanding of Dyna-Q**, seeing how it integrates planning with reinforcement learning to improve sample efficiency.
- **Explored the use of MPC**, applying model-based control strategies to optimize decision-making in dynamic environments.

Additionally, **a section on World Models will be introduced**, where you will investigate how agents can learn an internal representation of the environment to improve generalization and long-term planning. World Models represent a powerful paradigm in reinforcement learning, enabling agents to simulate experiences and plan actions more effectively.

This homework will not only strengthen your coding and analytical skills but also deepen your understanding of the interplay between model-based and model-free methods in reinforcement learning.

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: Monte Carlo Tree Search	40
Jupyter Notebook	28
Questions	12
Task 2: Dyna-Q Algorithms	40 + (4)
Jupyter Notebook	32 + (4)
Questions	8
Task 3: Model Predictive Control	20
Jupyter Notebook	15
Questions	5
Task 4: World Models (Bonus 1)	30
Jupyter Notebook	25
Questions	5
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 2: 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 1404/1/8 (March 21st 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 `HW5_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_HW5_[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 `HW5_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 Task 1: Monte Carlo Tree Search

1.1 Task Overview

This notebook implements a **MuZero-inspired reinforcement learning (RL) framework**, integrating **planning, learning, and model-based approaches**. The primary objective is to develop an RL agent that can learn from **environment interactions** and improve decision-making using **Monte Carlo Tree Search (MCTS)**.

The key components of this implementation include:

1.1.1 Representation, Dynamics, and Prediction Networks

- Transform raw observations into **latent hidden states**.
- Simulate **future state transitions** and predict **rewards**.
- Output **policy distributions** (probability of actions) and **value estimates** (expected returns).

1.1.2 Search Algorithms

- **Monte Carlo Tree Search (MCTS)**: A structured search algorithm that simulates future decisions and **backpropagates values** to guide action selection.
- **Naive Depth Search**: A simpler approach that expands all actions up to a fixed depth, evaluating rewards.

1.1.3 Buffer Replay (Experience Memory)

- Stores entire **trajectories** (state-action-reward sequences).
- Samples **mini-batches** of past experiences for training.
- Enables **n-step return calculations** for updating value estimates.

1.1.4 Agent

- Integrates **search algorithms** and **deep networks** to infer actions.
- Uses a **latent state representation** instead of raw observations.
- Selects actions using **MCTS, Naive Search, or Direct Policy Inference**.

1.1.5 Training Loop

1. **Step 1**: Collects trajectories through environment interaction.
2. **Step 2**: Stores experiences in the **replay buffer**.
3. **Step 3**: Samples sub-trajectories for **model updates**.
4. **Step 4**: Unrolls the learned model **over multiple steps**.
5. **Step 5**: Computes **loss functions** (policy, value, and reward prediction errors).

6. **Step 6**: Updates the neural network parameters.

Sections to be Implemented

The notebook contains several placeholders (`TODO`) for missing implementations.

1.2 Questions

1.2.1 MCTS Fundamentals

- What are the four main phases of MCTS (Selection, Expansion, Simulation, Backpropagation), and what is the conceptual purpose of each phase?
- How does MCTS balance exploration and exploitation in its node selection strategy (i.e., how does the UCB formula address this balance)?

1.2.2 Tree Policy and Rollouts

- Why do we run multiple simulations from each node rather than a single simulation?
- What role do random rollouts (or simulated playouts) play in estimating the value of a position?

1.2.3 Integration with Neural Networks

- In the context of Neural MCTS (e.g., AlphaGo-style approaches), how are policy networks and value networks incorporated into the search procedure?
- What is the role of the policy network's output ("prior probabilities") in the Expansion phase, and how does it influence which moves get explored?

1.2.4 Backpropagation and Node Statistics

- During backpropagation, how do we update node visit counts and value estimates?
- Why is it important to aggregate results carefully (e.g., averaging or summing outcomes) when multiple simulations pass through the same node?

1.2.5 Hyperparameters and Practical Considerations

- How does the exploration constant (often denoted c_{puct} or c) in the UCB formula affect the search behavior, and how would you tune it?
- In what ways can the "temperature" parameter (if used) shape the final move selection, and why might you lower the temperature as training progresses?

1.2.6 Comparisons to Other Methods

- How does MCTS differ from classical minimax search or alpha-beta pruning in handling deep or complex game trees?
- What unique advantages does MCTS provide when the state space is extremely large or when an accurate heuristic evaluation function is not readily available?

2 Task 2: Dyna-Q

2.1 Task Overview

In this notebook, we focus on **Model-Based Reinforcement Learning (MBRL)** methods, including **Dyna-Q** and **Prioritized Sweeping**. We use the Frozen Lake environment from Gymnasium. The primary setting for our experiments is the 8×8 map, which is non-slippery as we set `is_slippery=False`. However, you are welcome to experiment with the 4×4 map to better understand the hyperparameters.

Sections to be Implemented and Completed

This notebook contains several placeholders (`TODO`) for missing implementations as well as some markdowns (`Your Answer:`), which are also referenced in section 2.2.

2.1.1 Planning and Learning

In the **Dyna-Q** workshop session, we implemented this algorithm for *stochastic* environments. You can refer to that implementation to get a sense of what you should do. However, to receive full credit, you must implement this algorithm for *deterministic* environments.

2.1.2 Experimentation and Exploration

The **Experiments** section and **Are you having troubles?** section of this notebook are **extremely important**. Your task is to explore and experiment with different hyperparameters. We don't want you to blindly try different values until you find the correct solution. In these sections, you must reason about the outcomes of your experiments and act accordingly. The questions provided in section 2.2 can help you focus on better solutions.

2.1.3 Reward Shaping

It is no secret that Reward Function Design is Difficult in **Reinforcement Learning**. Here we ask you to improve the reward function by utilizing some basic principles. To design a good reward function, you will first need to analyze the current reward signal. By running some experiments, you might be able to understand the shortcomings of the original reward function.

2.1.4 Prioritized Sweeping

In the **Dyna-Q** algorithm, we perform the planning steps by uniformly selecting state-action pairs. You can probably tell that this approach might be inefficient. Prioritized Sweeping can increase planning efficiency.

2.1.5 Extra Points

If you found the previous sections too easy, feel free to use the ideas we discussed for the *stochastic* version of the environment by setting `is_slippery=True`. You must implement the **Prioritized Sweeping** algorithm for *stochastic* environments. By combining ideas from previous sections, you should be able to solve this version of the environment as well!

2.2 Questions

You can answer the following questions in the notebook as well, but double-check to make sure you don't miss anything.

2.2.1 Experiments

After implementing the basic **Dyna-Q** algorithm, run some experiments and answer the following questions:

- How does increasing the number of planning steps affect the overall learning process?
- What would happen if we trained on the slippery version of the environment, assuming we **didn't** change the *deterministic* nature of our algorithm?
- Does planning even help for this specific environment? How so? (Hint: analyze the reward signal)
- Assuming it takes N_1 episodes to reach the goal for the first time, and from then it takes N_2 episodes to reach the goal for the second time, explain how the number of planning steps n affects N_1 and N_2.

2.2.2 Improvement Strategies

Explain how each of these methods might help us with solving this environment:

- Adding a baseline to the Q-values.
- Changing the value of ε over time or using a policy other than the ε-greedy policy.
- Changing the number of planning steps n over time.
- Modifying the reward function.
- Altering the planning function to prioritize some state–action pairs over others. (Hint: explain how **Prioritized Sweeping** helps)

3 Task 3: Model Predictive Control (MPC)

3.1 Task Overview

In this notebook, we use MPC PyTorch, which is a fast and differentiable model predictive control solver for PyTorch. Our goal is to solve the Pendulum environment from Gymnasium, where we want to swing a pendulum to an upright position and keep it balanced there.

There are many helper functions and classes that provide the necessary tools for solving this environment using **MPC**. Some of these tools might be a little overwhelming, and that's fine, just try to understand the general ideas. Our primary objective is to learn more about **MPC**, not focusing on the physics of the pendulum environment.

On a final note, you might benefit from exploring the source code for MPC PyTorch, as this allows you to see how PyTorch is used in other contexts. To learn more about **MPC** and **mpc.pytorch**, you can check out OptNet and Differentiable MPC.

Sections to be Implemented and Completed

This notebook contains several placeholders (`TODO`) for missing implementations. In the final section, you can answer the questions asked in a markdown cell, which are the same as the questions in section 3.2.

3.2 Questions

You can answer the following questions in the notebook as well, but double-check to make sure you don't miss anything.

3.2.1 Analyze the Results

Answer the following questions after running your experiments:

- How does the number of LQR iterations affect the MPC?
- What if we didn't have access to the model dynamics? Could we still use MPC?
- Do `TIMESTEPS` or `N_BATCH` matter here? Explain.
- Why do you think we chose to set the initial state of the environment to the downward position?
- As time progresses (later iterations), what happens to the actions and rewards? Why