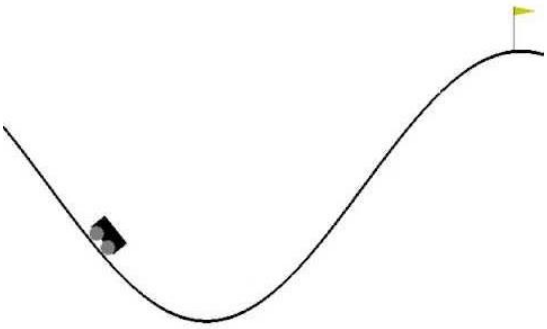
Ciências ULisboa | Informática

Advanced Machine Learning

PROJECT 2: Deep Reinforcement Learning

INTRODUCTION

Mountain Car is a classic reinforcement learning problem, often used as a benchmark for testing the performance of reinforcement learning algorithms. In this problem, an underpowered car must climb a steep hill to reach a goal located at the top of the hill. The car is subject to the laws of physics, which means that it cannot simply drive straight up the hill. Instead, it must build up speed by accelerating back and forth across the hill.



The state of the Mountain Car environment is represented by **two continuous variables**, the position and velocity of the car. The goal of the agent is to learn how to control the car's acceleration to climb the hill and reach the goal as quickly as possible while using the least amount of energy. **The agent receives a negative reward for every step it takes to reach the goal, so the goal is to minimise the number of steps required to reach the goal.**

OBJECTIVE

The main goal of this project is to **implement three agents to solve the Mountain Car problem**: the Q-learning and two deep learning agents (using deep Q-network and soft actor-critic). To accomplish this, some main sub-goals are underlying: understanding the algorithms to train one agent and its associated challenges. The deep learning agents are implemented in a JAX package to accelerate the training phase. However, performing all the experiments required to tune the parameters for the deep learning agents and training the Q-learning agent as implemented in classes can take some time, so **start working on the project as soon as possible**.

ENVIRONMENT

Before starting to work on the project, you should read the documentation about this problem in Gymnasium. There you can find the description of the problem, the observation space, the action space, and the reward. See this link. In the project, you will not work directly with the environment described in Gymnasium but with an accelerated version implemented in Gymnax (see this link).

WORK PLAN

This project has three main parts:

1. Train an agent with Q-learning (as done in TPs).
2. Train an agent with deep Q-network (DQN).
3. Train an agent with soft actor-critic (SAC).

To fulfil those three parts, you need to perform the following tasks:

1. **Train the Q-learning agent.**
    - To train this agent, you can use the Gymnasium environment version as it was done in classes.
    - Since the space is continuous, you need to discretise the position and velocity of the observation space for it to work with Q-tables. In this case, you will have a matrix of dimensions $P \times V \times A$, where P corresponds to the number of discrete positions, V to the number of discrete velocities and A to the number of actions. Be careful with the problem's dimensionality, as training will take longer.
    - You need to choose the parameters and implement the epsilon-greedy. You can decide to implement an epsilon decay as the episodes evolve.
    - You need to implement the Q-learning algorithm as learned in classes. Check the notebook from TP10 that explains how to do it.
    - You need to validate your agent measure, for instance, the total reward obtained in each episode and/or the number of steps in each episode required to complete the task. You may present the result as plots in function of the number of episodes performed.
2. **Learn deep reinforcement learning with JAX.** Study the notebook ("cartpole.ipynb") that explains how to use the library "RLinJAX" to train different deep learning agents, such as DQN and SAC. The notebook presents an example of the CartPole-v1 environment, which you must adapt for the project.
3. **Train the DQN agent.**
    - Implement a DQN agent using RLinJAX and train it to solve the problem. You might need to set some of the parameters.
    - You need to validate your agent so you can use the same measures as for Q-learning.
4. **Train the SAC agent.**
    - Study the paper "Soft Actor-Critic for Discrete Action Settings" by Petros Christodoulou (see this link) to understand the algorithm.
    - Implement a SAC agent using RLinJAX and train it to solve the problem. You are free to change the parameters you might consider relevant for training.

- Try different values for the algorithm-specific parameter "target_entropy_ratio" and study the influence of that parameter in the results.
- Again, you need to validate your agent so you can use the same measures as for Q-learning.

5. **Agents comparison.** Compare the three developed agents (Q-learning, DQN and SAC). The Gymnax library (see documentation in this link) presents a way to create gifs of the trained agents. Optionally, you can try to create gifs to see the difference between the agents.

SUBMISSION

Each group should submit the report written in Jupyter Notebook with the name **AAA2425_P2_xx.ipynb**, where the group number registered in Moodle should replace xx. The report should include (but not limited to):

- The identification of the group members (number and name of each element).
- Implementation of all agents (you need to describe all your experiments for each agent, what you did, the code to train the final agent and the results obtained).
- Comparison and discussion of the results obtained by the agents.

The report should also include your **explanations and justifications for your decisions**, as well as the **code and corresponding outputs** obtained.

Beware that **I will not run the code of all groups, but I might run some randomly selected groups,** so I need explicit outputs in the report to confirm your conclusions.

DEADLINE

The deadline for this project is **January 3rd at 23:59 in Moodle**.