Assignment 03: Learning for Control

Andrea Del Prete* - andrea.delprete@unitn.it

December 2023

1 Description

As final project for this course, students can choose between three options:

- A) Learning a Value function with a Neural Network, and using it as terminal cost in an optimal control problem.
- B) Learning the Viability kernel of a system and using it as terminal constraint in an MPC framework to ensure recursive feasibility.
- C) Learning a Value function, then learning a policy by minimizing the corresponding Q function, and finally using this policy to warm start an optimal control solver.

Contrary to the other assignments, this time you have to write all the code (almost) from scratch.

2 Submission procedure

You are encouraged to work on the assignments in groups of 2 people. **If you have a good reason to work alone, then you can do it, but this has to be previously validated by one of the instructors**. Groups of more than 2 people are not allowed. The mark of this assignment contributes to 25% of your final mark for the class (i.e. 7.5 points out of 30). The project discussion then contributes to another 20% (i.e. 6 points).

When you are done with the assignment, please submit a single compressed file (e.g., zip). **The file name should contain the surnames of the group members**, and it must contain:

- A pdf file with a detailed description of the work, the **names and ID number** of the group members; you are encouraged to include plots and/or numerical values obtained through simulations. **This pdf does not need to be long. Four to six pages of text should be enough. You can then add other pages for plots and tables.**
- The complete orc folder containing all the python code that you have developed.

If you are working in a group (i.e., 2 people) only one of you has to submit.

Submitting the pdf file without the code is not allowed and would result in zero points. Your code should be consistent with your answers (i.e. it should be possible to produce the results that motivated your answers using the code that you submitted). If your code does not even run, then your mark will be zero, so make sure to submit a correct code.

*Learning and Optimization for Robot Control, Industrial Engineering Department, University of Trento.

3 Common Tools

3.1 Solving Optimal Control Problems (OCPs)

All the three projects described above (and detailed in the following) begin by solving a series of OCPs. As an interesting alternative to using the single-shooting formulation that we have seen during our lab sessions, we suggest students to use CasADi[1]. CasADi is a software library for modeling and solving OCPs, coming with Python bindings. CasADi can be easily installed using pip:

```
pip install casadi
```

Alternative, you can use a modified version of the docker image of this course:

```
docker pull andreadelprete/orc23:casadi
```

An example of a script using CasADi to solve a simple OCP is available in `lab/orc/A3/casadi_example.py`.

When using CasADi, the system dynamics must be explicitly written down in the code. While the dynamics of a single pendulum is trivial, the dynamics of a double pendulum is rather complex. For this reason, we provide in the project folder a python script containing a function to evaluation the dynamics of a double pendulum.

3.2 Training Neural Networks

All projects require training neural networks. A template file containing code for creating and training a neural network using the *Tensor Flow/Keras* library is available in the folder `lab/orc/A3`. Installation instructions for tensor flow are in a text file located inside the same folder. Students are free to use other libraries for training neural networks, such as *PyTorch* or *JAX/FLAX*.

3.3 Parallel Computation

To make your code run faster you can try to parallelize it, so that it can exploit multiple CPU cores. To do that, you can use the Python library `multiprocessing`.

4 Project A — Learning a Terminal Cost

The aim of this project is to learn an approximate Value function that can then be used as terminal cost in an MPC formulation. The idea is similar to the one explored in the second assignment. First, we have to solve many OCP's starting from different initial states (either chosen randomly or on a grid). For every solved OCP, we should store the initial state x_0 and the corresponding optimal cost $J(x_0)$ in a buffer. Then, we should train a neural network to predict the optimal cost J given the initial state x_0. Once such a network has been trained, we must use it as a terminal cost inside an OCP with the same formulation, but with a shorter horizon (e.g. half). We should be able to empirically show that the introduction of the terminal cost compensates the decrease of the horizon length.

For this test, we suggest to start using a single pendulum, and then moving to a double pendulum.

[1] https://web.casadi.org/

5 Project B — Learning a Terminal Constraint Set

This project is the continuation of the second assignment. After generating the data that has been plotted in the second assignment, we have to train a neural network to classify states as either viable, or non-viable. Then, we can use such a network for constraining the terminal state of an MPC formulation to be viable. Thanks to this extra constraint, we should be able to show that the MPC problem remains recursively feasible in situations where feasibility would be lost without using such a constraint.

For this test, we suggest to start using a single pendulum, and then moving to a double pendulum. To make the problem challenging in terms of constraint satisfaction, we suggest to use a cost function that pushes the system to reach a joint configuration that is close to the joint limits, while penalizing very little the joint velocities and torques.

6 Project C — Actor-Critic Learning

The first part of this project is similar to project A. First, we have to solve many OCP's starting from different initial states (either chosen randomly or on a grid). For every solved OCP, we should store the initial state x_0 and the corresponding cost $V(x_0)$ in a buffer. Then, we should train a neural network (called "critic") to predict the cost V given the initial state x_0. After the critic has been trained, we should train another neural network, called "actor", that approximates the greedy policy with respect to the critic. To train this network we should minimize the action-value function corresponding to the critic, that is:

$$\pi(x) = \min_u \; l(x,u) + V(f(x,u)), \tag{1}$$

where l is the running cost, V is the critic, and f is the system dynamics. After training the critic, we can use it either to directly control the system, or to compute an initial guess (for both state and control) to provide to the OCP solver. In both cases we can measure the performance of the actor by evaluating the resulting cost of the OCP, and comparing it to the cost of the first batch of OCP's.

For this project, we suggest to start with a simple 1D single integrator, that has the following dynamics:

$$x_{t+1} = x_t + \Delta t \, u_t \tag{2}$$

To make the problem interesting (i.e. non-convex), you can use the following running cost:

$$l(x,u) = \frac{1}{2}u^2 + (x-1.9)(x-1.0)(x-0.6)(x+0.5)(x+1.2)(x+2.1) \tag{3}$$

Once things are working with this system, we can move to a more complex double integrator:

$$\begin{bmatrix} p_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_t \\ v_t \end{bmatrix} + \begin{bmatrix} 0.5\Delta t^2 \\ \Delta t \end{bmatrix} u_t \tag{4}$$

7 Suggestions

When preparing the final report, make sure to account for the following tips.

1. Mathematically describe the optimal control problem formulation.

2. Describe the structure of the neural network (e.g., number of layers, number of neurons per layer, activation functions).
3. Report the problem constraints.
4. Include plots of some trajectories (position, velocity, torque) obtained controlling the system with the optimal policy.
5. Report the value of the hyper-parameters of the algorithm (e.g., learning rate, mini-batch size, number of data points).
6. Report the training time.
7. Include plots (color maps) of the Value and policy function whenever possible.
8. Include videos of the robots controlled with the policy found.
9. Include a discussion of potential improvements you did not have time to implement/test.