universität freiburg

Model Predictive Control and Reinforcement Learning

Overview of the Exercise Sessions

Joschka Boedecker, Moritz Diehl, Andrea Ghezzi, Jasper Hoffmann

Exercise 1: Dynamic Systems and Simulation

In this exercise, we will get to know the software tool CasADi.
We consider a simple inverted pendulum. The system dynamics are given via the ODE:

$$\begin{aligned} \dot{\theta} &= \omega \\ \dot{\omega} &= \sin(\theta) + \tau, \end{aligned} \tag{1}$$

where θ is the angle describing the orientation of the pendulum, ω is its angular velocity and τ is the input torque.

1. **CasADi.** Check out the CasADi documentation[1]. Make sure you understand the difference between a CasADi expression and a CasADi function.
2. **From continuos-time to discrete-time.** We build the discrete time dynamic system from the continuous time ODE using one RK4 step.
3. **Differentiation.** By means of the CasADi function `jacobian` we obtain two CasADi functions that compute the Jacobian of the discrete time dynamics with respect to states and actions respectively.
4. **Simulation.** Complete the templated code to simulate the system forward in time starting from the initial state $s_0 = (\pi/2, 0)$ with constant input $a_k = 0$, $k = 0, \dots, 200$. What do you observe?

Exercise 2: Numerical Optimization and Optimal Control

In this exercise we focus on numerical optimization and numerical optimal control.
In the **first part**, we regard the following optimization problem

$$\begin{aligned} \min_{x,y} \quad & \frac{1}{2}(x-1)^2 + \frac{1}{2}(10(y-x^2))^2 + \frac{1}{2}x^2 \\ \text{s.t.} \quad & x + (1-y)^2 = 0 \end{aligned} \tag{2}$$

1. **KKT conditions.** Write down the KKT conditions for the above problem. Are these conditions necessary for optimality? Are they sufficient?
2. **CasADi.** Define the problem using CasADi.
3. **IPOPT.** Solve the problem via IPOPT and plot level lines, constraint and optimal solution.

In the **second part**, we bring together what we have learnt so far - discretization and simulation of dynamical systems and numerical optimization - to formulate an Optimal Control Problem (OCP). We consider the inverted pendulum defined in (1), we aim to compute a control trajectory such that the pendulum is stabilized in the upward position. To do so, we formulate the following OCP via direct multiple shooting

$$\begin{aligned} \min_{\substack{s_0, a_0, \dots, \\ a_{N-1}, s_N}} \quad & \frac{1}{2}\sum_{k=0}^{N} s_k^\top Q s_k + a_k^\top R a_k + \frac{1}{2} s_N^\top Q s_N \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{k+1} = F(s_k, a_k), \\ & h(s_k, a_k) \geq 0, \qquad k = 0, \dots, N-1, \\ & r(s_N) \geq 0, \end{aligned} \tag{3}$$

and we solve it again with IPOPT via CasADi.

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

1. **Dynamics.** Define the ODE of the system and create a RK4 integrator.
2. **Cost.** Define the stage cost and the terminal cost
3. **OCP.** Construct the OCP in CasADi by multiple shooting, create a instance of `nlpsol` and solve the OCP.
4. **Plot.** Retrieve the optimal solution and plot the state and action trajectories.

Exercise 3: Dynamic Programming and LQR

In this exercise, we will use dynamic programming (DP) to implement a controller for the inverted pendulum (1). The goal is to design a feedback policy capable of swinging up the pendulum starting from $\theta = \pi$. We express this as a discrete time optimal control problem:

$$\begin{aligned} \min_{\substack{s_0, a_0, \dots, \\ a_{N-1}, s_N}} \quad & \frac{1}{2}\sum_{k=0}^{N} s_k^\top Q s_k + a_k^\top R a_k + \frac{1}{2} s_N^\top Q_N s_N \\ \text{s.t.} \quad & s_0 = \bar{s}_0, \\ & s_{k+1} = F(s_k, a_k) \;\; k = 0, \dots, N-1, \\ & -10 \leq a_i \leq 10, \;\; k = 0, \dots, N-1, \\ & -\pi/2 \leq \theta_i \leq 2\pi, k = 0, \dots, N, \\ & -8 \leq \omega_i \leq 8, \quad k = 0, \dots, N, \end{aligned} \tag{4}$$

with $\bar{s}_0 = (\pi, 0)$.

1. **LQR.** Consider the unconstrained linear quadratic infinite horizon problem that is obtained by linearizing the dynamics at $s_{\text{lin}} = (0, 0)$ and $a_{\text{lin}} = 0$ and dropping the control constraints. Compute the LQR gain matrix K. Grid the state space and compute the corresponding LQR cost and control policy (impose control constraint via clipping).
2. **Dynamic Programming.** By following the templated code, implement the DP algorithm and use it to compute the cost-to-go associated with the initial state of the optimal control problem.
3. **Comparison.** Consider the plots provided by the previous template, showing the cost of DP and LQR as well as their control policies. Where is the LQR policy similar to the one obtained with DP? Where is it different? Why?

Exercise 4: Pytorch

In this exercise, we introduce the fundamental concepts of PyTorch and show how this framework can be applied to solve real-world research challenges.
PyTorch is a Python-based library designed for deep learning. It is distinguished by its dynamic computational graph, which enables researchers and developers to construct models with a high degree of flexibility. PyTorch has found extensive use in various scientific and engineering domains due to its ease of use and extensive research-friendly features. This exercise is based on the "PyTorch 60-Minute Blitz"[2]

1. **Tensors.** Learn how to deal with PyTorch "tensors", they are comparable to NumPy's ndarrays but they can run on GPUs and other accelerators.

[2] `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

2. **Autograd.** Autograd is PyTorch's engine for automatic differentiation and it is essential from training neural network. In the templated code you learn how to compute gradients of PyTorch expression defined with tensors.

3. **Neural Network with PyTorch.** Build your neural network model using the package `torch.nn`, you will see how to define the architecture and the loss function, and how to do backpropagation and update the weights.

4. **Training a classifier.** Bring together all you have learnt in the steps above to train a classifier based on Convolutional Neural Network (CNN) for discerning images from the CIFAR10 dataset.

Exercise 5: Tabular Q-Learning

In this exercise, we implement tabular Q-Learning and solve the "Cliff Walking" environment. We introduce the python package `gymnasium`[3], aka, `gym`, which provides an API to communicate between learned models and environments.

1. **Import environment.** Import the "Cliff Walking" environment from `gym` and learn the basic commands.

2. **Q-Learning.** Implement an ϵ-greedy action selection, and then, fill out the template to implement the Q-Learning algorithm. Run your routine and plot the learned policy.

3. **Convergence of Q-Learning.** Take a look to the training statistics via the provided plot functions.

Exercise 6: Deep Q-Learning

In this exercise, we implement DQN by Mnih et al.[4]. DQN was the first algorithm that was able to play Atari games while only using images as an input, showing the potential of deep reinforcement learning. We consider as example the pendulum on cart (or cart pole) and we aim to stabilize the pendulum in the upward position. For this example the possible number of states is too high to be able to store in a table.

1. **Q-function approximation.** Use a neural network to learn a parameterized Q-value function that can be applied to the whole state.

2. **Experience buffer and training.** Store the transitions that the agent experiences in "replay buffer", this enable us to reuse older experience. Use the collected experience for training the parameterized Q-value function.

3. **DQN.** Follow the templated code to implement the DQN algorithm.

4. **Deploy on Cart Pole.** Deploy the DQN algorithm you implemented on the Cart Pole environment.

[3] `https://gymnasium.farama.org/`
[4] https://arxiv.org/abs/1312.5602