

Robot Learning Project

December 19th, 2024

TAs: Andrea Protopapa (andrea.protopapa@polito.it), Luca Faieta (s323770@studenti.polito.it)

0 Introduction

The main goal of this project is to get familiar with the problem of learning a control policy for a robot in simulation using state-of-the-art reinforcement learning algorithms while introducing the challenges of using that policy in the real world. In particular, the student will learn about the *sim-to-real transfer* problem in robot learning literature, namely the task of learning policies in simulation through RL that can be directly transferred to real-world hardware, avoiding costly interactions with real setups and speeding up the training time. During this project, the student will be simulating the sim-to-real transfer task in a *sim-to-sim* scenario, where a discrepancy between *source* (training) and *target* (test) domains is manually injected. The student will implement *domain randomization* of dynamics parameters (e.g. masses, friction coefficients), a popular strategy to learn robust policies that transfer well to the target domain.

Specifically, the student must reach the following goals by the end of the project:

1. Read the provided material to get familiar with the Reinforcement Learning framework, the sim-to-real transfer challenge and the common techniques to perform an efficient transfer from simulation to reality;
2. Implement an RL training pipeline via third-party APIs to state-of-the-art reinforcement learning algorithms such as PPO and SAC.
3. Implement Uniform Domain Randomization (UDR) to learn robust policies in the source domain and limit the loss of performance during the sim-to-real transfer;
4. Propose and implement a novel extension of the project;

To proceed with the assignment, clone the starting code from the repository at the [dedicated webpage](#) and follow the steps listed below in this document.

Your submission should consist of (1) a **PDF report** (with paper-style) containing a **discussion of the results and answers to the questions** asked in these instructions, and (2) the **code** used for the exercise. In particular, the report should contain a brief introduction, a related work section, a methodological section, an experimental section with all the results and discussions, and a final brief conclusion. Remember to also describe your own project extension in the report. Follow this [link](#) to open and create the template for the report using Overleaf.

The **deadline to submit** the report and code **through the Github Classroom** is January 17th at 23:59 for those attending the exam at the first call, or January 29th at 23:59 for the second call.

1 Preliminaries

Before starting, you're asked to take some time to familiarize yourself with the framework of Reinforcement Learning, the sim-to-real transfer challenge and SOTA strategies to overcome it. More in detail:

- Read sections 1.-1.4, 1.6, 3.-3.8, of [1] to understand the general Reinforcement Learning framework;
- Watch introductory video on Reinforcement learning by DeepMind [video](#)
- Read [article](#) on introduction to Reinforcement learning by OpenAI [part 1, part 2, part 3]
- Read sections 1.-1.3, 3.-3.4, of [2]
- Read sections 1., 2., 3. of [3]
- Read debate on the sim-to-real transfer paradigm [4]
- Read [5], [6], [blog post](#) to understand domain randomization for sim-to-real transfer
- Read this [set of slides](#) by Josh Tobin and this [article](#) regarding domain randomization for both vision and dynamics properties

2 Implement lower/upper bound baselines

Train an RL agent on the gym [Hopper](#) environment. This environment comes with an easy-to-use python interface which controls the underlying physics engine — [MuJoCo](#) — to model the robot.

The hopper is a one-legged robot model whose task is to learn how to jump without falling, while achieving the highest possible horizontal speed.

The student will simulate the sim-to-real transfer scenario in a simplified sim-to-sim setting, as no work takes place on an actual real robot. In particular, two custom *domains* have been created ad-hoc: policy training takes place in the *source* environment and the student will transfer and test the policy on the *target* environment — which technically represents the real world. To simulate the reality gap, the source domain Hopper has been generated by shifting the torso mass by 1kg with respect to the target domain.

2.1 Hopper environment

Task 1 Check out the provided [code template](#) and start playing around with the underlying Hopper environment. Get familiar with the `test_random_policy.py` script, the python interface for MuJoCo, the [gym documentation](#), and the hopper environment overall. Finally answer the questions below.

Question 1.1 What is the state space in the Hopper environment? Is it discrete or continuous?

Question 1.2 What is the action space in the Hopper environment? Is it discrete or continuous?

Question 1.3 What is the mass value of each link of the Hopper environment, in the source and target variants respectively?

Hint: If you need any help answering the above questions try looking at the [Mujoco documentation](#) or the [gym documentation](#).

A few other hints:

- Bodies defined in the environment: `env.sim.model.body_names`
- Mass of all the corresponding bodies: `env.sim.model.body_mass`
- Number of degrees of freedom (DoFs) of the robot: `env.sim.model.nv`
- Number of DoFs for each body: `env.sim.model.body_dofnum`
- Number of actuators: `env.sim.model.nu`
- See other attributes [here](#)

2.2 Implement the RL training pipeline

Task 2 Implement a reinforcement learning pipeline to train a simple control policy for the Hopper environment. To this end, you'll make use of a third-party library to train an agent with state-of-the-art reinforcement learning algorithms such as PPO and SAC. In particular, follow the steps below, and make sure to go through the provided external resources:

1. Create a script using the third-party library [stable-baselines3](#) (sb3) and train the Hopper agent with **one** algorithm of choice between PPO [8] and SAC [7].
 - a. [openAI article on PPO](#)
 - b. [openAI article on SAC](#)
 - c. Explanation [video](#) on PPO, explanation [video](#) on SAC.
2. Use the provided template in `train.py` as a starting point. It is okay to look at publicly available code for reference, but it's likely easier and more helpful to study the sb3 documentation and understand how to implement the code by yourself.

2.3 Training and testing

Task 3 Train two agents with your algorithm of choice, on the *source* and *target* domains respectively. Then, test each model and report its average return over 50 test episodes. In particular, report results for the following “training→test” configurations:

- source→source,
- source→target (**lower bound**),
- target→target (**upper bound**).

Test with different hyperparameters and report the best results found together with the parameters used.

Question 3.1 Why do we expect lower performances from the “source→target” configuration w.r.t. the “target→target”?

Question 3.2 If higher performances can be reached by training on the target environment directly, what prevents us from doing so (in a sim-to-real setting)?

3 Uniform Domain Randomization

Implement Uniform Domain Randomization (UDR) for the link masses of the Hopper robot.

In this setting, UDR refers to manually designing a uniform distribution over the three remaining masses in the *source* environment (considering that the torso mass is fixed at -1 kg w.r.t. the target one) and performing training with values that vary at each episode (sampled appropriately from the chosen distributions).

The underlying idea is to force the agent to maximize its reward and solve the task for a range of multiple environments at the same time, such that its learned behavior may be robust to dynamics variations. Note that, since the choice of the distribution is a hyperparameter of the method, the student has to manually try different distributions in order to expect good results on the target environment.

Task 4 Train a UDR agent on the *source* environment with the same RL algorithm previously used. Later test the policy obtained on both the *source* and *target* environments.

Question 4.1 Is UDR able to overcome the unmodelled effect (shift of torso mass) and lead to more robust policies w.r.t. the naive “source→target” configuration in task 3?

Question 4.2 Can you think of limitations or downsides of UDR?

Hints:

- `env.sim.model.body_mass[i]` controls the mass of the *i*-th body in the Hopper environment. In particular, the torso mass value is `env.sim.model.body_mass[1]`, the thigh mass value is `env.sim.model.body_mass[2]`, and so on.

- To check out all body names: `env.sim.model.body_names`

- Remember not to randomize the torso mass!

4 Project extension

At this stage, you are expected to implement your own extension to the project. This phase may include the implementation of any idea of yours to further improve the sim-to-real transfer in our simple scenario—e.g. investigate the individual importance of randomizing the different masses (thigh, leg, foot)—but is not limited to this problem. For example, you are free to carry out a novel analysis on particular aspects of the reinforcement learning pipeline, which are not necessarily obvious—e.g. extend the analysis to more MuJoCo environments, thoroughly analyzing the impact of different network structures/configurations, etc. There are no particular constraints on the choice of your extension, as long as the idea/analysis is well motivated and technically sound when implemented.

Keep in mind that, rather than requiring you to obtain actual improvements, this step is for you to go beyond the project guidelines and get a feeling of a research-like approach.

References

- [1] "Reinforcement Learning: An introduction (Second Edition)" by Richard S. Sutton and Andrew G. Barto, [PDF](#)
- [2] Kober, J., Bagnell, J. A., & Peters, J. (2013). "Reinforcement learning in robotics: A survey". The International Journal of Robotics Research, [PDF](#)
- [3] Kormushev, P., Calinon, S., & Caldwell, D. G. (2013). "Reinforcement learning in robotics: Applications and real-world challenges", [PDF](#)
- [4] Höfer, S., Bekris, K., Handa, A., Gamboa, J. C., Golemo, F., Mozifian, M., ... & White, M. (2020). "Perspectives on sim2real transfer for robotics: A summary of the R: SS 2020 workshop", [PDF](#)
- [5] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World." arXiv, Mar. 20, 2017. [PDF](#)
- [6] Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018, May). "Sim-to-real transfer of robotic control with dynamics randomization", [PDF](#)
- [7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.", [PDF](#)
- [8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). "Proximal policy optimization algorithms", [PDF](#)