Sim-to-Real transfer of Reinforcement Learning policies in robotics

TA: Gabriele Tiboni (gabriele.tiboni@polito.it)
Link to this file: shorturl.at/a1246

Advanced Machine Learning 2022
Politecnico di Torino

OVERVIEW

The main goal of this project is to get familiar with the reinforcement learning (RL) paradigm and apply it in the context of robot learning. To this end, the student will be introduced to the problem of learning a control policy for a robot in simulation using state-of-the-art reinforcement learning algorithms and methods. 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 recent strategy to learn robust policies that transfer well to the target domain. Finally, the student will also attempt to train control policies directly from visual input (i.e. images) through convolutional neural networks, allowing the robot to autonomously interact with the environment from raw images as observations.

GOALS

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. Study the code and implement an RL training pipeline via third-party APIs to state-of-the-art reinforcement learning algorithms such as PPO, TRPO, 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. Train vision-based control policies from raw input images

1st STEP) STUDY BACKGROUND & RELATED WORKS

Getting familiar with the reinforcement learning framework in robotics

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:

- 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

2nd STEP) IMPLEMENTING AND TESTING DIFFERENT ALGORITHMS FOR REINFORCEMENT LEARNING

Defining the baseline/upper bound for the next phase

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 a transfer scenario on this robot, given two custom variants which 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.

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 in the table below.

State space	**Action space**	**Mass values**
What is the state space in the Hopper environment? Is it discrete or continuous?	What is the action space in the Hopper environment? Is it discrete or continuous?	What is the mass value of each link of the Hopper environment, in the source and target variants respectively?

If you need any help answering the above questions try looking at the Mujoco documentation or the gym documentation.
A few hints:

- *Name of all bodies defined in the env: env.model.body_names*
- *Number of degrees of freedom (DoFs) of the robot: env.model.nv*
- *Number of DoFs for each body: env.model.body_dofnum*
- *Number of actuators: env.model.nu*

2. Implement a reinforcement learning pipeline to train a 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 TRPO, PPO, and SAC. In particular, follow the steps below, and make sure to go through the provided external resources:
    a. Create a script using the third-party stable-baselines3 (sb3) library and train the Hopper agent with **one** algorithm of choice between TRPO [8], PPO [9] and SAC [7].
        i. openAI article on TRPO
        ii. openAI article on PPO
        iii. openAI article on SAC
        iv. Explanation video on TRPO and PPO, explanation video on SAC.

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.

3. Train two agents with your algorithm of choice, on the source and target domain Hoppers respectively. Then, test each model and report its average reward 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:

Table 1) **RL algorithm**	**Hyperparameters**	**Source-Source average return**	**Source-Target average return**	**Target-Target average return**
<TRPO, PPO, SAC>	*e.g., learning rate: 1e-3, training episodes: 100000*			

The results above will serve as the upper bound and lower bound for the Domain adaptation phase using domain randomization.

3rd STEP) IMPLEMENTING DOMAIN RANDOMIZATION

Implement Uniform Domain Randomization to narrow the gap between source and target domains and make your agent more robust to different environment dynamics.

Implement Uniform Domain Randomization (UDR) on the mass of each link of the Hopper.

Uniform domain randomization (in the proposed setting) 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 true one) and doing 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 slight 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.

1. 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. Finally, report their average reward in the Table below.

   Is UDR able to overcome the unmodelled effect (shift of torso mass) and lead to more robust policies?

   Suggestions:
   - env.model.body_mass[1:] controls the mass of the four links in the Hopper environment. In particular, the torso mass value is env.model.body_mass[1].
   - remember not to randomize the torso mass!

Table 2) **RL Algorithm**	**Hyperparameters**	**UDR distribution**	**Source-Source average return**	**Source-Target average return**
<TRPO, PPO, SAC>	*E.g., learning rate: 1e-3, training episodes: 100000*	*e.g.* *[1.5kg, 3kg]* *[2kg, 3kg]* *[2kg, 3kg]*		

4th STEP) VISION-BASED REINFORCEMENT LEARNING

Implement an RL training pipeline that uses raw images as state observations to the agent. In this final step, you're given more flexibility on how you can implement the pipeline, leaving room for variants and group's own implementations.

1. Implement your own new training script so as to change the observations of the environment to only use raw images (i.e. 2D renders) of the environment rather than low-dimensional configuration vectors, and to use a convolutional neural network of your choice as the underlying policy structure.

   Train an agent on raw images with your RL algorithm of choice and test it on source-source, source-target, target-target configurations.

Suggestions:

- Have a look at the gym.wrappers documentation, and specifically at PixelObservationWrapper().

- Stacking images from previous timesteps often helps to keep information on the velocity of moving objects! See [14] for more details and the FrameStack gym wrapper for an easy implementation.

Table 3) **RL Algorithm**	**Hyperparameters**	**UDR distribution**	**Source-Source average return**	**Source-Target average return**
<TRPO, PPO, SAC>	*E.g., learning rate: 1e-3, training episodes: 100000*	*e.g.* *[1.5kg, 3kg]* *[2kg, 3kg]* *[2kg, 3kg]*		

2. **(optional)** Variants: at this stage, you may feel free to implement any idea to attempt at further improving the sim-to-real transfer in our simple scenario, for the vision-based RL task.
   For example, variants may try to investigate domain randomization on the appearance of the images to improve the transfer from source to target domain. Other possible directions may include investigating better representation learning techniques (such as transfer learning from pre-trained convolutional neural networks) or domain augmentation.
   Rather than requiring you to obtain actual improvements, this step is for you to go beyond the guidelines and get a feeling of a research-like approach.

AT THE END

- Deliver PyTorch scripts for all the required steps.
- Deliver this file with the tables compiled.
- Write a complete PDF report (with paper-style). The report should contain a brief introduction, a related work section, a methodological section for describing the algorithm that you're going to use, an experimental section with all the results and discussions, and a final brief conclusion. Follow this **link** to open and create the template for the report.

EXAMPLE OF QUESTIONS YOU SHOULD BE ABLE TO ANSWER AT THE END OF THE PROJECT

- What is Reinforcement Learning?
- Why is Reinforcement Learning appealing for the robotics field?
- Why are physics simulators popularly used to learn Reinforcement Learning policies for real robots?
- What is the task given in the Hopper environment?
- What is the reward function in the Hopper environment?
- What are the challenges of the sim-to-real transfer paradigm?
- What is the reality gap?
- What are the popular strategies for performing an efficient transfer?
- What is Domain Randomization?
- What is Uniform Domain Randomization?
- What are the limitations of UDR?
- What is vision-based RL?

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., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). “Trust region policy optimization”, PDF
[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). “Proximal policy optimization algorithms”, PDF
[10] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., & Fox, D. (2019, May). “Closing the sim-to-real loop: Adapting simulation randomization with real world experience”, PDF
[11] Tiboni, G., Arndt, K., & Kyrki, V. (2022). “DROPO: Sim-to-Real Transfer with Offline Domain Randomization”, PDF
[12] Tsai, Y. Y., Xu, H., Ding, Z., Zhang, C., Johns, E., & Huang, B. (2021). “Droid: Minimizing the reality gap using single-shot human demonstration”, PDF
[13] Muratore, F., Eilers, C., Gienger, M., & Peters, J. (2021). “Data-efficient domain randomization with bayesian optimization”, PDF
[14] V. Mnih et al., “Playing Atari with Deep Reinforcement Learning.” arXiv, Dec. 19, 2013. PDF