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

Michele Morese
Politecnico di Torino
s302182@studenti.polito.it

Abstract

Nowadays, one of the main problems of the Reinforcement Learning paradigm is its challenging application to robotics, due to the difficulties of learning in the real world and modeling physics into simulations. In this report, a method of Domain Randomization is explored, Uniform DR, to find out how it deals with former methods' issues and show its strengths and weaknesses.

1. Introduction

As the aim of the project is to analyze different proposed approaches to the Sim-to-Real problem in the field of Reinforcement Learning, in this section the main concepts of this learning paradigm is described. The code of the implemented methods can be found in the project repository [5].

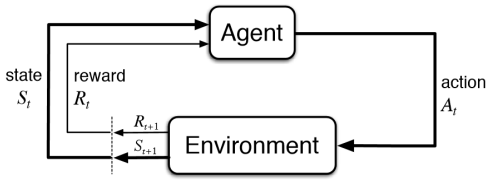

Figure 1. General Reinforcement Learning scheme

1.1. Reinforcement Learning

The general learning framework used throughout the project is Reinforcement Learning (RL), in which the knowledge is acquired by experience. The main objective of Reinforcement Learning is to train an agent that maximize a numerical reward, R_t. The agent observes the environment's current state, S_t, which is used to determine the actions, A_t, to performe. The actions are taken following a particular strategy, called policy $\pi(\cdot)$. The general schema is summarized in Fig. 1.
As already said, RL deals with learning from experience. The agent first experiments random actions, than, taking into account the rewards obtained, is able to determine the action that would lead to an higher reward, avoiding those that results to poor performances. How much the agent experiments with new random actions or follow the results previously obtained is defined through the learning rate. In this report, we will analyze a well known algorithm of RL, in particular, a policy-gradient method. We will talk more in detail about the algorithm in Section 2.
We introduce now some common quantities that will be used in the next sections:

- The sequence of $S_0, A_0, \ldots, A_{T-1}, S_T$ is called episode and it finishes when the S_T is a terminal state.
- The return at a given time t:

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=t+1}^{\inf} \gamma^{k-t-1} R_k$$

  is the sum of rewards starting from time t weighted by powers of the discount factor $\gamma \in [0, 1]$. The discount factor determine how much into the future we care about.
- The state-value function $V_\pi(S)$ of a state S under a policy π is the expected return obtained by starting in S and following π.
- The policy performance measure $J(\theta)$ with respect to the parameters θ of the policy π_θ. It is the expected return starting from the initial state S_0 by following the policy π_θ. It holds that $J(\theta) = V_{\pi_\theta}(S_0)$.

1.2. Sim-to-Real

Reinforcement Learning can be used to solve very complex tasks and thus it lends itself to being a good learning framework for robots. However, the process of training could be very long and expensive in a real world setup and could also bring dangerous or misleading outcomes. One solution for solving these issues is to speed up the training procedure using a simulator, building an approximate model of the environment dynamics. However, it's not possible for the simulator to predict all the different condition

possible in the real world. So, a reality gap is defined as the discrepancy between the real world dynamics and the simulation. The existence of this gap indicates that a model is able to learn only the approximate dynamics of the simulator, leading to poor performances in a real environment. The challenge of correctly transferring the experience from the simulation to the real world is called Sim-to-Real, and it deals with training policies that are more likely to adapt to the real environment, while still trained in a different one.
Inside the Sim-to-Real world, we will focus on Domain Randomization (DR) [6], a methodology that allows a more robust policy, obtained through training the model in different conditions. This is possible thanks to random variations of the environment parameters, generating a series of slightly different environments on which the agent can be trained on. An increase in the amount of different conditions that the agent learns how to face results in a decreased probability of overfitting the simulation. In this report, we will deal with one versions of it: Uniform DR in Section 3.

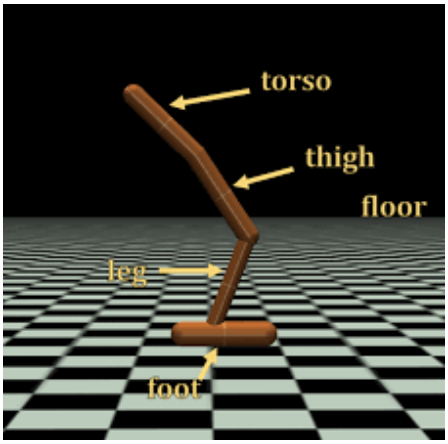

Figure 2. Hopper environment

1.3. Simulation setup

The environment taken into consideration is the Hopper environment of the Gym API [3], which consists of a two-dimensional one-legged robot composed of four main body parts: torso, thigh, leg and foot (see Fig. 2). The goal of the Hopper is to learn how to make hops in the forward direction, without falling down, by applying a torque on the three hinges that connects the different parts. The hopper is considered unhealthy when it falls down or its configuration does not allow it to remain in balance; otherwise, it is considered healthy.
In our setting:

- The environment is a simulated flat world containing the robot. For the source environment, the body parts of the robot have the following masses: 2.53 kg for torso, 3.93 kg for thigh, 2.71 kg for leg and 5.09 kg for foot. The target environment, instead, differs only for the mass of the torso, which is 3.53 kg.
- The agent controls the robot, based on its current state. It is the one controlling the training process of the policy π_θ, represented by a Multi-Layer Perceptron:
  - The input layer has the dimensions of the State Space, including the quantities observed by the agent at each state (coordinates, angles of joints, velocities).
  - The output layer has the dimensions of the Action Space, that is a set of the three torques that generate the action.
  - θ is a parameterization of the policy that coincides with the weights and biases of the neural network.
- The reward is composed of 3 parts:
  1. Healthy Reward: a reward of +1 for each time step of life.
  2. Forward Reward: at each time step T a reward of hopping forward, measured as $(x_{t+1} - x_t)/\Delta t$, where Δt is the duration of the action in time steps. This rewards indicates that longer step that maintains the hopper healthy are preferred.
  3. Control Cost: a negative reward for penalizing the hopper if it takes actions that are too large. It acts as a regularization term for learning smooth trajectories without sudden accelerations.

  The total reward returned is defined as HealthyReward + ForwardReward - ControlCost.

2. Reinforcement Learning Algorithm

In this section the algorithm employed during the project is described. The algorithm used is characterized by a policy gradient method, which is a technique that rely upon optimizing parameterized policies by gradient descent. For the algorithm a pseudocode will be presented along with the results obtained.

2.1. PPO

Proximal Policy Optimization (PPO) [1, 4] is a method for optimizing stochastic control policies using the concept of Trust Region. If the policy is performing good, the train procedure should perform the updates in such a way that the new policy has a behavior similar to the old one. This can be expressed in terms of KL-divergence, which is a measure of distance between distributions.
There are two primary variants of PPO: PPO-Penalty and PPO-Clip.

- PPO-Penalty approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the course of training so that it's scaled appropriately.
- PPO-Clip doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

In this project, the stable-baselines3 [2] implementation of the algorithm has been used, in particular, the Clip version (the primary variant used at OpenAI).

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

The model has been first optimized, performing a simple grid search considering different combinations of parameters, shown in Table 1. Each combination has been trained for 200,000 timesteps, and the best one has been highlighted in the same table.

Table 1. Search Space for PPO

Hyper Parameter	Search Space
learning rate	{3e-4, **2.5e-4**, 1e-3}
batch size	{**64**, 128}
n steps	{1024, **2048**}

2.2. Results

The best combination of parameters found in the previous section has been trained for 3,000,000 timesteps both on the source and target environment. The model obtained training on the source has been tested on both the environment for 50 episodes, while the model trained on the target has been tested only on the target environment,

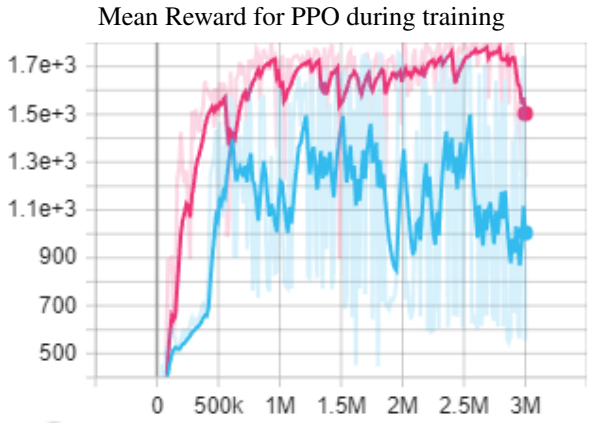

Figure 3. Comparison between the behaviour of the models tested on the source (blue) and target (red) environments

Figure 4 shows the mean reward obtained by the models trained on the source and target enviroment. The results have been smoothed by a factor of 0.8 through the tensorboard struments, while the raw data can be seen with a lighter color. It can be seen that the model trained on the target environment obtain higher and more stable results, while training on the source returns a more unstable model, even if it can reach the same best results in few episodes.

Both the models have been tested on differents environments, with the results shown below:

Table 2. Average reward during testing

Source-Source	Source-Target	Target-Target
1519.99	552.46	1797.93

As expected, the model trained on the target environment perform better than the one trained on source when tested on the same environment as the training. Additionally, it can be highlighted the high drop in performance when the Source model is tested on the target environment. This drop represent the difficulty of the model in adapting when facing a different environment: this is the reality gap that was previously presented in Section 1.2 The following section is focused on solving these issues by means of a Domain Randomization technique.

3. Uniform Domain Randomization

3.1. Description

As introduced in section 1.2, Domain Randomization is a method that tries to compensate the incapability of the

simulator on representing all the complex effects of the real world, by varying the training environment at each episode.

3.2. Implementation

Since the two environments differ only for the mass of the torso, the parameters to be randomized will be the masses of the other body parts. Uniform Domain Randomization associates an uniform distribution U(a_i, b_i) to each parameter, from which their values are sampled at each episode. Algorithm 2 shows a simple pseudocode of the method.

Algorithm 2 UDR

Require: $\{m_i\}_{i=1}^{d}$ {Parameters to be randomized}
Require: $\{(a_i, b_i)\}_{i=1}^{d}$ {Distribution bounds}
  repeat
    for $i \in d$ **do**
      $m_i \sim U(a_i, b_i)$
    end for
    Generate Data (m_i)
    Update policy
  until Training is complete

3.3. Results

The algorithm has been tested with 8 different randomized environments using different widths for the parameters' distributions, defined as [mean value - width, mean value + width]. The mean value represent the weight of the body part in a non-randomized environment.
In Table 3 is reported the search space for these parameters.

Table 3. Search Space for UDR

Parameter width	Search Space
thigh	{0.5, **0.75**}
leg	{0.5, **0.75**}
foot	{**0.5**, 0.75}

Each combination of parameters' width has been trained for 200,000 timesteps on the source environment and tested on the same. The best combination, highlighted in the table, was able to achieve a mean reward of 1571.48.

These parameters have been used to train the model on the source environment for 5,000,000 timesteps, with the results shown in Figure 4. The results have been smoothed by a factor of 0.7 through the tensorboard struments, while the raw data can be seen with a lighter color. It can be seen that the model trained using the UDR technique on the source environment returns a more stable behaviour, despite providing similar performance, with respect to the model presented in section 2.2. Also, it's possible to highlight how the model slowly returns higher/lower rewads overtime, that can be explained by the need to adapt to different environments. A longer training could lead to a stable performance in all the different possible environments.

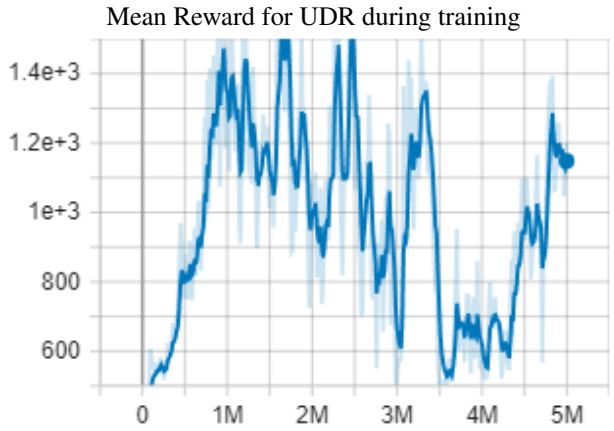

Figure 4. Smoothed behavior of the model tested on the source environment

The model have then been evaluated for 50 episodes on both source and target environment, with the results shown in Table 4:

Table 4. Average reward during testing

Source-Source	Source-Target
1657.93	1070.01

As expected, the model still present a drop in performance when tested on the target environment. However, when compared with the results obtained with the PPO model and presented in section 2.2, it's possible to see the improvement of the model when confronted with an environment different to the training one.
This comparison confirms that the UDR technique permits to close the reality gap faced by other algorithm. However, the drop in performance could still be too high to be acceptable, and a further improvement could be necessary.

4. Coclusion

During the project, one of the main Reinforcement Learning algorithms was studied, observing the poor performances when adapting to a slightly different environment. Therefore, the Sim-to-Real problem was addressed with Domain Randomization, testing the implementation of

UDR. UDR is a simple algorithm that helps to improve the performance of Sim-to-Real. However, one main problem is that it requires a big effort on the manual tuning. Also, due to the high randomization of the environments since the beginning of the training, the model requires a long time to adapt to all the different conditions. A last criticism is the reality gap that the model still present, being unable to perfectly adapt to the variable condition that the model could face in the real world.

References

[1] "https://spinningup.openai.com/en/latest/algorithms/ppo.html" PPO OpenAI article. 2

[2] Antonin Raffin et al. "stable-baselines3: Reliable reinforcement learning implementations". 2021. 3

[3] Greg Brockman et al. "openai gym". 2

[4] John Schulman et al. "proximal policy optimization algorithms". 2017. 2

[5] Michele Morese. Sim to real transfer of reinforcement learning policies in robotics, 2023. https://github.com/Alakebada/daai2022-2023-p4rl. 1

[6] X. B. Peng, M. Andrychowicz, W. Zaremba, and P Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. May 2018. 2