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

Ahmadreza Farmahini Farahani[1], *Ramin Hedayatmehr*[1], *Masoud Karimi*[1]

***Abstract*—Reinforcement learning(RL) is a computational approach to understanding and automating goal-directed learning and decision making, RL emphasises on learning by an agent from direct interaction with its environment, without requiring exemplary supervision or complete models of the environment. RL is defined as interactions between a learning agent and its environment in terms of states, actions, and rewards. The agent's objective is to maximize the amount of reward it receives over time. In this project, model-free RL with policy optimization are adopted. In robotics, it is often unrealistic to assume that the true state is completely observable and noise-free. The reality gap between source and target domain, form the barrier to using simulated data on real robots. Domain randomization is a simple but promising method for addressing the reality gap. In this project randomization is done on physical parameters such as robot joints.**

I. Introduction

The main aim of this project is trying to train an RL agent on the gym Hopper environment [1]. The hopper is a one-legged robot model that consist of four main body parts whose task is to learn how to jump without falling; The goal is to make hops that move in the forward (right) direction by applying torques on the three hinges connecting the four body parts. An action represents the torques applied between links. Observations consist of positional values of different body parts of the hopper, followed by the velocities of those individual parts (their derivatives) with all the positions ordered before all the velocities.

The reward consists of three parts:

- *healthy_reward:* Every timestep that the hopper is healthy, it gets a reward of fixed value
- *forward_reward:* This reward would be positive if the hopper hops forward (positive x direction).
- *ctrl_cost:* A cost for penalising the hopper if it takes actions that are too large.

To do that, underlying physics engine — MuJoCo [2] — is used to model the robot. MuJoCo is a free and open source physics engine that aims to facilitate research and development in robotics, biomechanics, graphics and animation, and other areas where fast and accurate simulation is needed.

II. Reinforcement Learning

A. Definition

RL is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, which evolutionary methods do not do, without requiring exemplary supervision or complete models of the environment. Reinforcement learning uses the formal framework of Markov decision processes to define the interaction between a learning agent and its environment in terms of states, actions, and rewards. Action space could be discrete or continuous. The actions are the choices made by the agent, the states are the basis for making the choices; and the rewards are the basis for evaluating the choices. Basically it's a number that tells how good or bad the current world state is [3].

B. Traditional RL

A policy is a rule used by an agent to decide what actions to take. It can be deterministic $a_t = \mu(s_t)$ or it may be stochastic $a_t \sim \pi(.|s_t)$. In another words, policy is a map from state to action [3].

A trajectory τ is a sequence of states and actions in the world. Actions come from an agent according to its policy.

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

The very first state of the world, s_0, is randomly sampled from the start-state distribution, sometimes denoted by ρ_0:

$$s_0 \sim \rho_0$$

The reward function is one of the important concepts in RL. The reward function R is critically important in reinforcement learning. It depends on the current state of the world, the action just taken, and the next state of the world, although frequently this is simplified to just a dependence on the current state [3].

$$r_t = R(s_t, a_t, s_{t+1}) \tag{1}$$

There are two kind of reward function, but the important one is infinite-horizon discounted return. which is the sum of all rewards ever obtained by the agent but discounted by how far off in the future they're obtained. This formulation of reward includes a discount factor $\gamma \in (0, 1)$. Actually γ means that agent cares more about the rewards that are give recently versus the same rewards that was given in the earlier time steps from now [3].

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2}$$

Whatever the choice of return measure (whether infinite-horizon discounted, or finite horizon undiscounted), and whatever the choice of policy, the goal in RL is to select a policy which maximizes expected return when the agent acts

[1] *Polytechnic Of Turin University*

according to it. If we suppose that both the environment transitions and the policy are stochastic. In this case, the probability of a T-step trajectory is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t_0=1}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{3}$$

The expected return denoted by J(π) is:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{4}$$

Therefor, the central optimization problem in RL can then be expressed by:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{5}$$

It's often useful to know the value of a state, or state-action pair. By value, we mean the expected return if you start in that state or state-action pair, and then act according to a particular policy forever after. This value can be obtained by value functions. There are four value functions; The Optimal Value Function, $V^*(s)$, which gives the expected return if you start in state s and always act according to the optimal policy in the environment:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)\,|s_0 = s] \tag{6}$$

All four of the value functions [3] obey special self-consistency equations called Bellman equations. The basic idea behind the Bellman equations is this: The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next. The Bellman equations for the optimal value functions are

$$V^*(s) = \max_{a} \mathbb{E}_{s'\sim P} r(s, a) + \gamma V^*(s') \tag{7}$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s, a)$, indicating that the next state s' is sampled from the environment's transition rules. Sometimes in RL, we don't need to describe how good an action is in an absolute sense, but only how much better it is than others on average. That is to say, we want to know the relative advantage of that action. We make this concept precise with the advantage function.

There are two kind of RL algorithms [4], Model-Free and Model-Based. The main upside to having a model is that it allows the agent to plan by thinking ahead, seeing what would happen for a range of possible choices, and explicitly deciding between its options. Agents can then distill the results from planning ahead into a learned policy. The main downside is that a ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to learn the model purely from experience, which creates several challenges. The biggest challenge is that bias in the model can be exploited by the agent, resulting in an agent which performs well with respect to the learned model, but behaves sub-optimally (or super terribly) in the real environment.

In this branch of RL we can learn:

- policies, either stochastic or deterministic
- action-value functions (Q-functions),
- value functions,
- environment models.

C. Deep RL

1) Actor-Critic: Actor-critic is a popular class of reinforcement learning algorithms used to learn an optimal policy for an agent in an environment with continuous state and action spaces. It combines the benefits of both policy-based and value-based methods, where the policy is represented by an actor, and the value function is represented by a critic.

The actor is responsible for selecting actions based on the current state of the environment, and the critic evaluates the goodness of the actor's actions. The actor-critic algorithm is an online, model-free, and on-policy algorithm, meaning that it learns the optimal policy from experience without any prior knowledge of the environment's dynamics.

The actor-critic algorithm works by simultaneously updating the actor and the critic using two separate loss functions. The actor's loss function is based on the advantage function, which measures how much better the current action is compared to the average action. The critic's loss function is based on the temporal difference error, which measures the difference between the estimated value of the current state and the actual value obtained from the next state.

2) Option-Critic Network: The option-critic network comprises three neural networks: an actor network, a critic network, and an option network. The actor network selects the option to be executed, the option network selects the primitive action to be executed, and the critic network evaluates the value of the current state and option.

Option-Critic Network, is a reinforcement learning algorithm that combines options and deep reinforcement learning to learn a hierarchical policy in environments with long horizons and complex action spaces. The option framework allows the agent to learn a set of temporally extended actions, or options, which can be executed repeatedly until a specific goal is achieved.

III. Simulation To Reality

A. Intrinsic Of Simulation To Reality

Simulation to reality(Sim2real) is a term used to describe the process of transferring policies learned in simulation to the real world. In this context, simulation refers to the use of virtual environments to train and evaluate machine learning models before deploying them in the real world. While simulations offer several advantages [5], they also have some disadvantages that can impact the effectiveness of sim2real transfer.

1) Advantages of simulation in sim2real:

- Cost-effectiveness: Simulations can be run at a fraction of the cost of real-world experiments, making it possible to generate large amounts of data and train models more efficiently.

TABLE I
GRID SEARCH RESULTS

	$\log(Learning\ rate) = -3$			$\log(Learning\ rate) = -4$			$\log(Learning\ rate) = -5$		
	$S = 1024$	$S = 2048$	$S = 4096$	$S = 1024$	$S = 2048$	$S = 4096$	$S = 1024$	$S = 2048$	$S = 4096$
$Clip\_range = 0.1$									
$< source, source >$	1563.26	1611.32	1616.63	473.384	1091.26	1256.27	250.207	401.677	191.471
$< source, target >$	744.236	1078.32	769.139	520.289	1179.15	1267.11	344.992	430.718	195.962
$< target, target >$	1526.29	1706.47	1727.18	592.898	1409.58	825.243	189.377	209.282	459.266
$Clip\_range = 0.2$									
$< source, source >$	1653.53	1541.6	1592.25	1060.18	174.04	1268.85	184.709	189.558	347.978
$< source, target >$	875.294	1589.72	833.81	1031.94	183.693	1388.49	515.449	304.079	486.951
$< target, target >$	1534.01	1716.06	1461.96	1253.17	1344.71	1302.29	234.351	223.509	289.405
$Clip\_range = 0.3$									
$< source, source >$	1461.96	1446.27	1502.46	976.682	1179.75	793.014	451.438	462.598	318.725
$< source, target >$	1427.41	891.222	925.176	1040.91	1221.33	1219.85	511.106	399.363	334.805
$< target, target >$	1339.21	1514.32	1531.16	1226.52	1236.73	1342.19	294.097	374.153	416.496

- Safety: Simulations provide a safe and controlled environment where agents can learn and explore without the risk of injury or damage to equipment.
- Control over environment: Simulations allow for precise control over environmental factors, such as lighting, weather, and other variables, which can be difficult or impossible to control in the real world.

2) Disadvantages of simulation in sim2real:

- Transfer ability issues: Even if a policy performs well in simulation, it may not transfer well to the real world due to differences in physics, sensors, and other factors.
- Lack of complexity: Simulations can be simplified versions of the real world, which may not fully capture the complexity of real-world environments and tasks.
- Reality Gap: The reality gap refers to the difference between the simulated environment and the real-world environment, which can make it difficult to transfer policies learned in simulation to the real world.

B. Solution To sim2real Problems

Bridging the 'reality gap' that separates simulated robotics from experiments on hardware could accelerate robotic research through improved data availability; Moreover, discrepancies between physics simulators and the real world make transferring behaviors from simulation challenging [6].

System identification, the process of tuning the parameters of the simulation to match the behavior of the physical system, is time-consuming and error-prone. Even with strong system identification, the real world has unmodeled physical effects like nonrigidity, gear backlash, wear-and-tear, and fluid dynamics that are not captured by current physics simulators.

if the variability in simulation is significant enough, models trained in simulation will generalize to the real world with no additional training. By randomizing the dynamics of the simulator during training, we are able to develop policies that are capable of adapting to very different dynamics, including ones that differ significantly from the dynamics on which the policies were trained.

IV. METHODS

A. Model

Unlike popular Q-learning approach which in that agent interacts with environment with policy π, it will store the transition experience (s, a, r, s') in replay buffer to store past experiences. On the other hand, in policy gradient algorithm, like Proximal Policy Optimization(PPO) [7], agent learns directly from the its experience in the environment which means that it's model-free RL. Thus, policy gradient loss function becames:

$$L^{PG}(\theta) = \mathbb{E}_t[log_{\pi_\theta}(a_t|s_t)\hat{A}_t]$$

In the equation, π_θ is our policy, in the logarithm part, probability of different actions in given state are calculated, while in the advantage estimate $\hat{A}_t$, calculates the relative value of selected action in the current state. To calculate value of advantage function, two terms must be calculated, first, discounted sum of rewards (return) and baseline estimate. Return, actually is weighted sum of all the rewards that agent got during each time step in the current episode. Afterwards, advantage is calculated when all the sequence of episodes are collected from environment, therefor all the rewards are known. The second part, baseline or value function, basically what the value function tries to do is give a estimate of discounted sum of rewards from this point on-wards. This term tries to guess what is the final return is going to be in this episode starting from current state. It's supervised learning because it calculates the return from this state on-wards. Since our networks may not be able to calculate the exact value of that state, thus, we will end up with noisy estimate of value function. Finally, the advantage estimate is, value of return minus baseline estimate, advantage estimation answer this question that the action done by agent is better that it was expected or worse. Thus, if $\hat{A}_t$ if positive the probability of action increases and the gradient will be positive, otherwise it will be decreased and consequently the gradient will be negative. One of problem is that is we keep running gradient descent on one batch of collected experience, actually we

TABLE II
TOP 10 RESULTS IN GRID SEARCH

log(Learning rate)	Gamma	Clip range	Steps	GAE lambda	$< source, source >$	$< source, target >$	$< target, target >$
0.001	0.99	0.2	4096	0.96	1697.19	969.815	1494.06
0.001	0.99	0.1	2048	0.97	1665.51	1134.21	1595.06
0.001	0.99	0.2	4096	0.97	1663.16	1690.15	1610.42
0.001	0.99	0.2	2048	0.97	1655.41	866.579	1591.16
0.001	0.99	0.2	1024	0.95	1653.53	875.294	1534.01
0.0001	0.99	0.1	4096	0.97	1649.48	797.617	1357.66
0.001	0.99	0.3	4096	0.97	1628.52	1603.18	1331.82
0.001	0.99	0.1	4096	0.96	1626.76	887.365	1595.96
0.001	0.99	0.1	4096	0.95	1616.63	769.139	1727.18
0.001	0.99	0.1	2048	0.95	1611.32	1078.32	1706.47

update the parameters in our network far from outside of range where data was collected, this leads to have a noisy estimated advantage function. To solving this issue we use be sure that we do not move far from the old policy, our objective function is divided by old policy:

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\ \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right), \quad (8)$$

the second term is clipped version of normal policy, in which ϵ is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.

B. Training

Hyperparameters play a crucial role in the RL algorithms. RL algorithms learn from interaction with an environment, and hyperparameters affect how this interaction takes place. Hyperparameters are settings that are determined before the learning process starts and are not learned during the training process. These parameters can significantly impact the performance of the algorithm. The choice of hyperparameters can determine whether the algorithm converges to an optimal policy or value function, and how quickly it does so. Therefore, selecting the right hyperparameters is crucial for achieving good performance in reinforcement learning tasks.

1) grid search: It involves creating a grid of possible values for each hyperparameter and then evaluating the performance of the algorithm for each combination of hyperparameters in the grid. The combination that results in the best performance is then chosen as the optimal set of hyperparameters. Grid search can be a time-consuming process, especially if there are many hyperparameters and possible values to consider. However, it is a straightforward and systematic approach that can help to identify good sets of hyperparameters for a given reinforcement learning task. In the Table. I, we can see results within grid search approach. In this table, discount factor ($gamma$) and factor for trade-off of bias vs variance for Generalized Advantage Estimator (gae_lambda) has been set to $gamma = 0.99$, $GAElambda = 0.95$. We also tried other values for discount factor and lambda but the empirical values of the stable baselines 3 are more promising. In the Fig. 1 we can see reward progress during training the hopper. Choosing appropriate learning rate is crucial for the application. Due to the high number of parameters it is not possible with grid search approach. In the future steps we tune this parameter. In the Fig. 2 we can see the effectiveness of choosing right gamma and trade-0f factor (gae lambda). As we can see choosing this two parameters are not critical as learning rate.

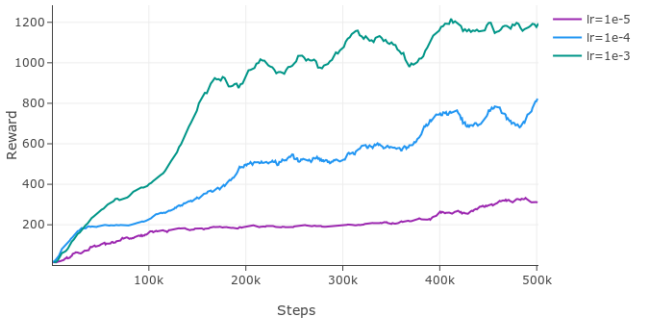


Fig. 1. Rewards regarding learning rate

We can see that the rewards on target are not consistent with the rewards on source. After training on target environment we can obtain better results on target, but in the real world we do not have access the target environment.

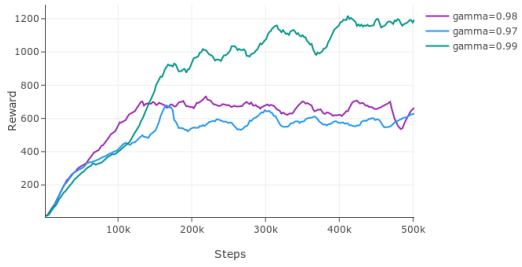
Now we consider top ten results for grid search. These models are obtained with training on source. After we trained and observed the results on source, we trained them on target environment. In the Table. II we have the results. As we have seen in the previous results, the rewards on target environment are not consistent with the rewards on source.

In Fig. 3 we have different traininng process with different values of clip range and steps (number of steps before updating the weighgts).

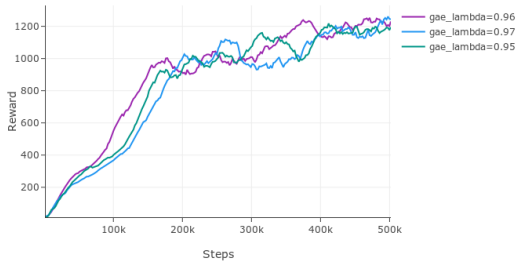
2) Optuna: Optuna uses a combination of pruning algorithms and Bayesian optimization to efficiently search for the best set of hyperparameters for a given machine learning task. Pruning algorithms enable Optuna to quickly discard unpromising hyperparameter configurations, while Bayesian optimization helps to balance exploration and exploitation in the search for the optimal configuration. Optuna can help to save time and improve the performance of machine learning

TABLE III
TOP 10 RESULTS IN OPTUNA

log(Learning rate)	Gamma	Clip range	Steps	GAE lambda	$< source, source >$	$< source, target >$	$< target, target >$
0.00054	0.989125	0.135039	1472	0.966356	1789.73	1286.98	1633
0.000544	0.988273	0.121027	1856	0.966962	1770.17	782.76	1528.31
0.00043	0.987974	0.1328	2112	0.960129	1734.72	690.81	1666.93
0.000523	0.991254	0.119511	2048	0.964195	1729.15	980.38	1632.01
0.000422	0.991761	0.125875	3904	0.959133	1728.83	902.68	1482.18
0.000569	0.990486	0.120183	3776	0.962096	1711.85	862.89	1620.91
0.000613	0.990919	0.106105	2240	0.964751	1697.85	1448.97	1655.23
0.000485	0.990101	0.107607	1472	0.963952	1695.92	1060.13	1484.92
0.000535	0.989854	0.118027	1408	0.967968	1687.01	734.97	1583.68
0.000408	0.990761	0.100542	1920	0.958415	1674.93	920.30	1682.3

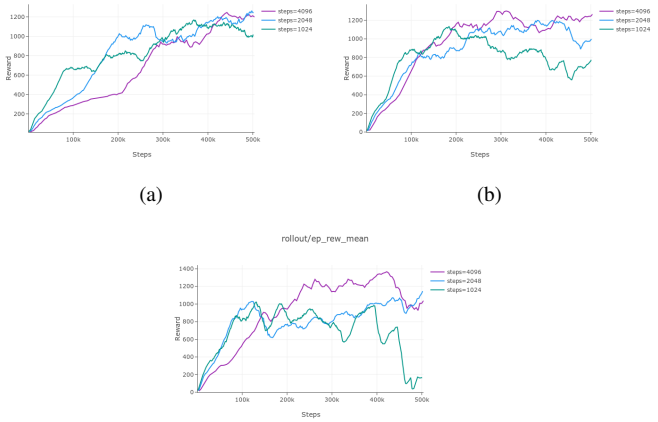


(a)



(b)

Fig. 2. (a) Rewards regarding to gamma (b) Rewards regarding to trade-off factor (gae lambda)



(a) (b)

(c)

Fig. 3. (a) Rewards regarding to steps with $Clip\_range = 0.1$ (b) Rewards regarding to steps with $Clip\_range = 0.2$ (c) Rewards regarding to steps with $Clip\_range = 0.3$

TABLE IV
DOMAIN RANDOMIZATION ON BEST MODEL

Upper bound	Lower bound	$< source, source >$	$< source, target >$
1.5	4	1563.44	1686.58
4	7.5	1589.33	1340.04
4.5	6	1175.56	1266.69
1	2	1400.21	1225.29
4.5	7.5	1541.71	1207.19

models by automating the tedious and time-consuming process of hyperparameter tuning.

In the Table. III we have the top 10 results within optuna tuning method. We can a significant improvement on source. In this method we use random parameters with pruning strategy that allow us to find optimum hyperparameters. We have still bad results within target environment.

For the next steps we try to optimize the top 10 results obtained with optuna. Furthermore, as we expected the models are too sensitive to hyperparameters.

V. DOMAIN RANDOMIZZATION

The problem of transferring control policies from simulation to the real world can be viewed as an instance of domain adaptation(DA) or domain randomization(DR). DA refers to a set of transfer learning techniques developed to update the data distribution in sim to match the real one through a mapping or regularization enforced by the task model. On the other Hand, thanks to DR we are able to create a variety of simulated environments with randomized properties and train a model that works across all of them. Both DA and DR are unsupervised. Compared to DA which requires a decent amount of real data samples to capture the distribution, DR may need only a little or no real data. Training happens in the source domain. We can control a set of randomization parameters in the source domain with a configuration, sampled

from a randomization space.

Discrepancies between source and target domain refer to differences and inconsistencies that exist between source and the data or context that the information is being transferred to target. By randomizing physical parameters such as robot joints, we can train policies in simulation for robot movement, and later transfer the policies directly to the Hooper robot without requiring additional fine tuning on the physical system.

if the variability in simulation is significant enough, models trained in simulation will generalize to the real world with no additional training. Unfortunately, discrepancies between physics simulators and the real world make transferring behaviors from simulation challenging. Bridging the 'reality gap' that separates simulated robotics from experiments on hardware could accelerate robotic research through improved data availability.

Training a policy on an ensemble of dynamics models can make the controller robust to modeling error and improve transfer to a real robot. Our objective is to train policies that can perform a task under the dynamics of the real world. At the start of each episode, a random set of dynamics parameters are sampled according to uniform distribution and held fixed for the duration of the episode. The parameters which we randomize include:

- the thigh in the middle,
- the leg in the bottom,
- single foot on which the entire body rests

During randomization the torso mass is kept invariable, different distributions in order to expect good results on the target environment are tested. In the future steps we have upper bound and lower bound of the distributions that has been determined with a grid search approach. In this method we try to use different distributions with different length (difference between lower bound and upper bound) to find the optimum distribution.

We start to apply domain randomization in a single model. In the Table. IV we try to apply domain randomization on best model in the previous stages (first model in Table. III). The rewards on the target has been significantly improved. But, In some cases, we have better results on target and bad results on source. We want to choose a model that is consistant with source and target.

In the Table. V we have the results of the best models obtained during optuna stage. We can see the improvement in target reward. In some cases we obtain better reward on source without randomization. We have trade off between these two rewards, our main goal is improving target reward.

VI. Vision based reinforcement learning

Vision-based reinforcement learning (RL) is a sub-field of artificial intelligence (AI) that combines computer vision techniques with reinforcement learning algorithms. It involves training an agent to learn optimal behavior by interacting with an environment using visual input.

In traditional RL, the agent receives state information from the environment, which is usually represented as a set of numerical features or a high-dimensional vector. However, in vision-based RL, the agent perceives the environment through visual observations, typically in the form of images or video frames.

The main challenge in vision-based RL is to extract meaningful information from raw visual input and use it to make decisions. This requires the agent to learn visual representations that capture relevant features and patterns in the environment. Various computer vision techniques, such as convolutional neural networks (CNNs), are commonly used to process visual input and extract useful features. The overall process of vision-based RL involves the following steps:

- Perception
- Feature extraction
- Action selection
- Reward feedback
- Learning

Perception: The agent receives visual observations from the environment, such as images or video frames.

Feature extraction: Computer vision techniques, like CNNs, are used to process the visual input and extract relevant features.

Action selection: The agent selects an action based on the extracted features and its learned policy. The policy can be represented by a deep neural network or another RL algorithm.

Reward feedback: The agent receives a reward signal from the environment based on the action taken. The reward signal guides the agent's learning process by reinforcing desirable behaviors and discouraging undesirable ones.

Learning: The agent updates its policy based on the reward signal and the observed state-action pairs. This is typically done using RL algorithms, such as Q-learning, policy gradients, or actor-critic methods.

By iteratively going through these steps, the agent learns to improve its decision-making process and maximize the cumulative reward over time. Vision-based RL has been successfully applied to various domains, such as robotics, autonomous driving, and video games.

It is worth mentioning that vision-based RL is a computationally demanding task due to the high dimensionality of visual input. Training an RL agent with raw visual input can require substantial computational resources and training time. However, recent advances in deep learning and hardware acceleration have made significant progress in addressing these challenges and improving the capabilities of vision-based RL agents.

A. Convolution Neural Network (CNN)

Convolution Neural Network (CNN) has been making brilliant achievements. It has become one of the most representative neural networks in the field of deep learning. Computer vision based on convolutional neural networks has enabled people to accomplish what had been considered impossible in the past few centuries, such as face recognition, autonomous

TABLE V
DOMAIN RANDOMIZATION ON TOP 10 RESULTS

log(Learning rate)	Steps	Clip range	GAE lambda	Gamma	Lower bound	Upper bound	Before randomization		After randomization	
							$< s, s >$	$< s, t >$	$< s, s >$	$< s, t >$
0.00054	0.989125	0.135039	1472	0.966356	1	4.5	1789.73	1286.98	1563.44	1686.58
0.000544	0.988273	0.121027	1856	0.966962	4	6.5	1770.17	782.76	1627.63	1665.45
0.00043	0.987974	0.1328	2112	0.960129	4	5.5	1734.72	690.81	1620.83	1669.6
0.000523	0.991254	0.119511	2048	0.964195	4	5.5	1729.15	980.38	1609.24	1666.39
0.000422	0.991761	0.125875	3904	0.959133	3	5	1728.83	902.68	1635.19	1721.19
0.000569	0.990486	0.120183	3776	0.962096	4.5	5.5	1711.85	862.89	1595.71	1651.44
0.000613	0.990919	0.106105	2240	0.964751	4	5.5	1697.85	1448.97	1618.66	1648.81
0.000485	0.990101	0.107607	1472	0.963952	4.5	7	1695.92	1060.13	1596.73	1630.2
0.000535	0.989854	0.118027	1408	0.967968	3.5	5.5	1687.01	734.97	1634.56	1668.53
0.000408	0.990761	0.100542	1920	0.958415	3.5	6.5	1674.93	920.30	1680.83	1710.58

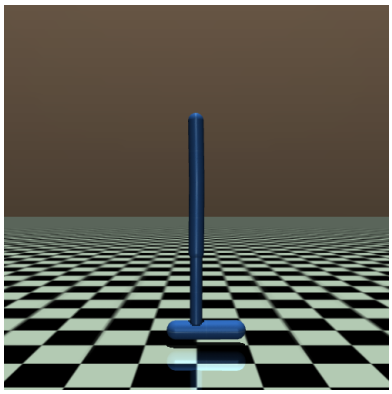


Fig. 4. Hopper rgb observation

vehicles, self-service supermarket, and intelligent medical treatment.

B. Pre-processing

For the development of this project, we changed the observation input to RGB pixels. This Input has a size of $500 * 500$. One option is to use each state as an observation of three channels. There are few problems with using the state RGB observations. In the Fig. 4 we can see the pixels space.

1. The RGB pixels format has three channels which each pixels indicate the velocity of red, green and red. But for our application, the colors are not important. Therefore, we have to reduce the number of channels. We use **Gray scaling**.

2. Large size of inputs. We have too many number of pixels. If if we reduce the size of the image, we have efficiency in time and parameters. We **Resize** the observation.

3. Each state values in this application is related to it's previous states and after. For example, we may have an observation but we do not know whether the hopper is falling or rising. We use **Stacking**.

4. High number of low value features in the pixels. The only feature that is valuable for us and has deterministic role in predicting the action, is the hopper itself. Meanwhile we have features in the pixels that their existence is not necessary. We use **Feature extraction** for this problem.

In the future hyperparameter tuning we use simpler approach than grid search. We just analyze the behaviour of each parameter regarding to a specific hyperparameter set.

C. Gray scaling

For reducing the number of channels, we change the RGB observation space to gray scale space. So we reduce the number of channels from three channels to one. That reduce the complexity of the model and training time. Since our application should work in real time, we should lower the complexity as much as possible. In the Fig. 5 we can observe the gray scale observation.

TABLE VI
CNN REWARDS REGARDING DIFFERENT INPUT IMAGE SIZES

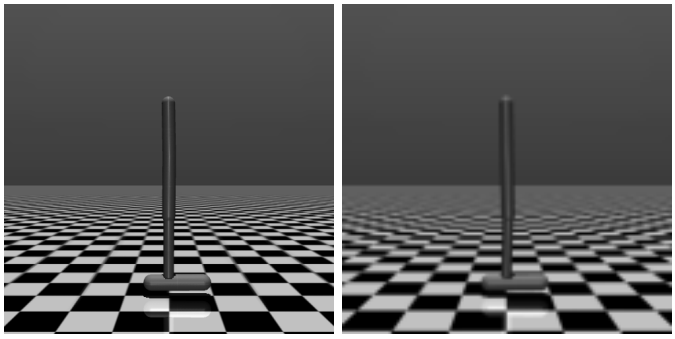
	$< s, s >$	$< s, t >$
	$n_stacks = 4$	
Base ($size = 500 * 500$)	140.12	138.54
Base ($size = 356 * 356$)	177.85	173.20
Base ($size = 244 * 244$)	176.35	167.69
Base ($size = 120 * 120$)	305.68	275.75

D. Resize

In the next step we try the resize the gray pixels to a lower dimensional space. Since choosing the right number of pixels for resizing is a hyperparameter, we should use different sizes to find the optimum value. In the Table. VI we can see the reward after resizing with different sizes. In the Fig. 5 we can see the resized observation along side the original size. Main features that can be used for training are still available in resized figure.

E. Stacking

Since we want to have information about previous states of an observation, we try to stack the current observation with the previous ones. We choose the right number of stacks by tuning it as a hyperparameter. In the Table. VII we can see the rewards obtained within different number of stacks.



(a) (b)

Fig. 5. (a) Gray scaled - original size (b) Gray scaled - resized

TABLE VII
CNN REWARDS REGARDING DIFFERENT NUMBER OF STACKS

	$< s, s >$	$< s, t >$
	$size = 120 * 120$	
Base ($n_stacks = 4$)	305.68	275.75
Base ($n_stacks = 8$)	174.27	166.16
Base ($n_stacks = 12$)	143.63	158.52
Base ($n_stacks = 16$)	108.06	110.57

F. Feature extraction using supervised technique

As we discussed before, for each frame before stacking, we want to extract meaningful features. We use two supervised and unsupervised methods for feature extraction. In this section we discuss about supervised version. We try to

TABLE VIII
CNN REWARDS WITH UNSUPERVISED PRE-PROCESSING

	$< s, s >$	$< s, t >$
Unsupervised pre-processing		
Base ($n_stacks = 4$, $size = 120 * 120$)	305.68	275.75
without unsupervised pre-processing		
Base ($n_stacks = 4$, $size = 120 * 120$)	305.68	275.75
Base ($n_stacks = 8$, $size = 120 * 120$)	174.27	166.16

set thresholds to delete not useful features. In this approach we discover thresholds with a supervised approach (since the pixels space for source and target is similar).In the Fig. 6 we can see the pre-processed space. We have compared the results of this approach and other scores we obtained so far in Table. VIII.

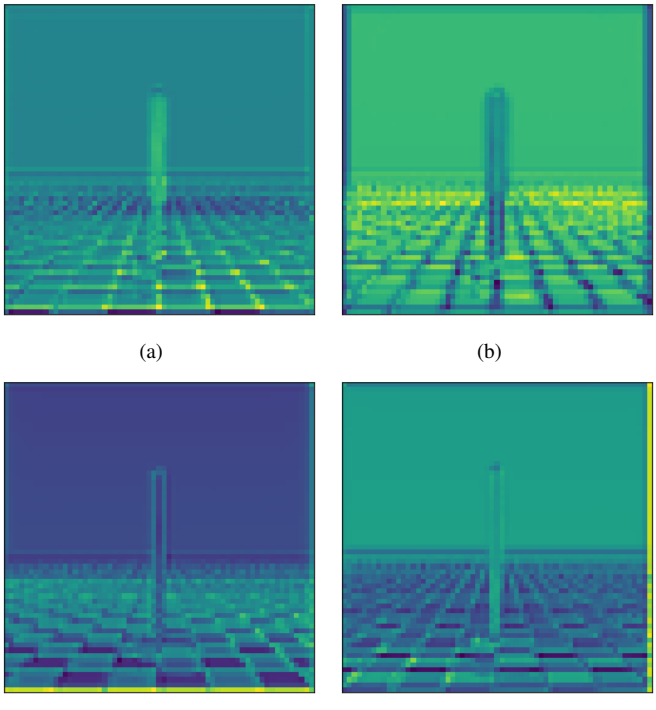
G. Feature extraction using transfer learning (unsupervised)

Transfer learning is a machine learning technique where a pre-trained model is used as a starting point for a new task. It leverages knowledge from the pre-trained model and adapts it to the new problem, typically with a smaller dataset. By reusing learned representations, transfer learning improves efficiency and performance, particularly when data is limited.

Fig. 6. Hopper supervised feature extraction

TABLE IX
CNN REWARDS WITH SUPERVISED AND UNSUPERVISED PRE-PROCESSING

	$< s, s >$	$< s, t >$
Supervised pre-processing		
MobilenetV2 ($n_stacks = 4$, $size = 120 * 120$)	180.60	171.86
Unupervised pre-processing		
Base ($n_stacks = 4$, $size = 120 * 120$)	135.56	184.60
Both supervised and unsupervised methods		
MobilenetV2 ($n_stacks = 4$, $size = 120 * 120$)	183.24	190.22
without supervised pre-processing		
Base ($n_stacks = 4$, $size = 120 * 120$)	305.68	275.75
Base ($n_stacks = 8$, $size = 120 * 120$)	174.27	166.16



(a) (b)

(c) (d)

Fig. 7. Features extracted from mobilenetV2

TABLE X
CONCLUSION RESULTS

	$< s, s >$	$< s, t >$
Raw state observation		
Optuna (best performance)	1680.83	1710.58
Pixels state observation		
Base [without pre-processing] ($n_stacks = 4$, $size = 120 * 120$)	305.68	275.75
Base [with pre-processing]($n_stacks = 4$, $size = 120 * 120$)	135.56	184.60
MobilenetV2 [without pre-processing]($n_stacks = 4$, $size = 120 * 120$)	180.60	171.86
MobilenetV2 [with pre-processing]($n_stacks = 4$, $size = 120 * 120$)	183.24	190.22

1. ResNet has two-layer residual block constructed by the shortcut connection. ResNet can mitigate the gradient vanishing problem without degeneration in deep neural networks since the gradient can directly flow through shortcut connections. Based upon ResNet, many studies have managed to improve the performance of the original ResNet, such as pre-activation ResNet , wide ResNet , stochastic depth ResNets (SDR) and ResNet in ResNet (RiR).

2. MobileNet v2

Based upon MobileNet v1, MobileNet v2 mainly introduces two improvements: inverted residual blocks and linear bottleneck modules. In MobileNet v2, an inverted residual block is opposite to a residual block. The input of an inverted residual block is firstly convoluted by 11 convolution kernels for channel expansion, then convoluted by 33 depth-wise separable convolution, and finally convoluted by 11 convolution kernels to compress the number of channels back. The whole process of an inverted residual block is channel expansion—depth-wise separable convolution—channel compression.

Also, due to the fact that depth-wise separable convolution cannot change the number of channels, which causes the number of input channels limits the feature extraction, inverted residual blocks are harnessed to handle the problem. When performing the steps: channel expansion—depth-wise separable convolution—channel compression, one problem will be encountered after "channel compression".

For the less complexity of the models we use MobileNet v2 as the base feature extraction model. In the Table. IX we can see the results obtained by this method.

Moreover, in the Fig. 7 we can visualize features extracted from mid-layers of MobilenetV2. Features are still have error in detecting the hopper but since it's an unsupervided method, it can be more useful than the supervised one. Computation is also faster.

We also try to develop a model with combination of both supervised and unsupervised method. The computions cost is a lot for this model and we use it only for experiment.

VII. CONCLUSION

After having bad results within target environment, we used domain randomization. We conclude that domain randomization can significantly improve the rewards on the target environment. With this technique, we simulated our training that can describe a real environment better.

Moreover, we tested our beliefs about the main reinforcement learning concepts. We observed the importance of hyperparameters. In classification tasks, there are convergence on complexity and good parameters but in the RL algorithms, convergence between different trainings are not possible due to the different data collection experiment. Each time we train an RL model, we are dealing with new data.

In the Table. X we can see the final results with different methods. All the models are domain randomized. We can conclude that using raw state observation with MLP has significantly better results than using pixels observation.

REFERENCES

[1] F. Foundation. Hopper agent. [Online]. Available: https://gymnasium.farama.org/environments/mujoco/hopper/

[2] D. T. Limited. Introduction mujoco. [Online]. Available: https://mujoco.readthedocs.io/en/latest/overview.html

[3] O. R. 038665d6. Key concepts in rl. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[4] ——. Kinds of rl algorithms. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[6] 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," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[7] O. R. 038665d6. Proximal policy optimization. [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/ppo.html#quick-facts

[8] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[9] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[10] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[11] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[8]–[11]