Uniform Domain Randomization in across-embodiment transfer learning

Giacomo Caciagli
Politecnico di Torino
Turin, Italy
giacomocaciagli@gmail.com

***Abstract*—The domain randomization is a powerful tool to help the sim-to-real transfer of reinforcement learning policies in robotic manipulation, it is a particular training method born to make policies more robust in order to achieve good performances in unseen scenarios. The characteristics of the domain randomization allow its use also in the field of across-embodiment transfer learning.**

This paper will show the results of the transfer of an agent, trained with a uniform domain randomization, in completely different environments. In particular, an agent created with the Proximal Policy Optimization algorithm in the hopper environment will be transferred in two more complex environments: the walker2d and the half cheetah environments.

***Index Terms*—reinforcement learning, domain randomization, transfer learning**

I. Introduction

The domain randomization is a newly method born to mitigate the problem of the sim-to-real transfer, the idea behind the domain randomization is to build a policy using a simulated environment where the data of the robot are not uniquely defined, which means that at each training episode some of the characteristics of the robot (that could be: weights or lengths of the links, starting position of the joints, starting position of the robot, etc...) change randomly. This training build a policy less effective but more robust, enough robust to overcome the so called sim-to-real transfer problem, a problem in which a policy created in a simulated environment doesn't perform well in the real world due to imprecise data about the real robot or the real environment.

A policy built with the domain randomization has learnt a behaviour that won't be linked to a specific environment, this specific characteristic of the policy makes it suitable to improve the results achievable in the across-embodiment transfer learning field.

The across-embodiment transfer learning is the transfer of a policy, obtained in any way, to a different agent in a new, and unseen, environment.

In this paper i will discuss the results of the domain randomization approach applied in the across-embodiment transfer learning, the experiments were made using Gym [1], a standard API for reinforcement learning, and three different MuJoCo environments: the hopper, the walker2d and the half cheetah (Fig. 1). The transfer was done from the hopper environment to the other two, the two target environments were chosen in such a way to be both more complex (with more degrees of freedom (DoF)) than the source environment and to have two different types of behaviour with different grades of similarity with respect to the source environment.

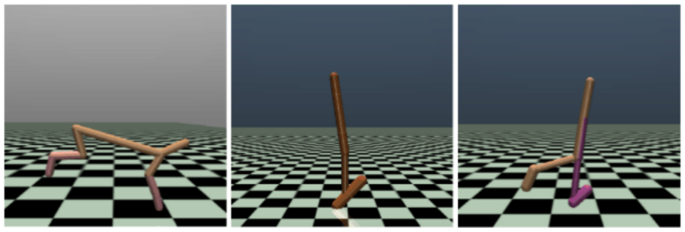
Fig. 1. The three environments, from left to right:
the half cheetah: 9 links that compose 8 body parts: tip, torso, back thigh, back shin, back foot, front thigh, front shin and front foot connected by 8 joints whose 2 fixed;
the hopper: 4 body parts: torso, thigh, shin and foot, connected by 3 joints;
the walker2d: 7 body parts: torso, right thigh, right shin, right foot, left thigh, left shin and left foot, connected by 6 joints;

The creation of all the policies was done using the Proximal Policy Optimization (PPO) algorithm, to see the impact of the domain randomization, two different policies were transferred, one trained in the standard way and the other trained with a uniform domain randomization (UDR), a particular case of the domain randomization where the characteristics of the robot change randomly following a uniform distribution.

II. Related work

The across-embodiment transfer learning problem has been highly studied and discussed in recent years but a unique and satisfying solution has not been found yet. Many solutions were found but are strongly dependent to the purpose of the robot and to the data which the robot is working with. For the type of transfer that i am working with, the most common solution is to train a policy in an environment, to use it as an initialization of the policy of the target environment and then to apply some finetuning in order to adapt it to the new environment. My work follows this path but the training phase is done with the domain randomization, due to the novelty of this method there are no similar studies to compare with.

III. Method

A. Proximal policy optimization algorithm

All the policies, for both the source and the targets environments, were created following the Proximal Policy Optimization (PPO) algorithm [2].

PPO is an on-policy algorithm that can be used with either discrete or continuous action spaces, it follows an actor-critic method, this means that the created agent is composed by two parts, as the name of the approach suggests, the actor and the critic, the actor is responsible to define and follow the policy while the critic must estimate the value function and evaluate the actions taken by the actor during the training phase, the two parts are trained together to achieve the highest reward possible. In order to properly read the results of the experiments is important to say that, due to the possibility of having episodes of infinite duration or extremely short, the duration of the training phase is not based on the number of episodes but is defined by the minimum amount of timesteps that the simulation must last, this means that the training will end only if the sum of the timesteps of all the episodes will be greater or equal than the decided amount.

B. Agent network

The agent created by the PPO algorithm is a neural network whose structure is defined by the Multilayer Perceptron (MLP) model, the high level scheme of the MLP model for all the MuJoCo environments is shown in Fig. 2. The network is composed by a total of seven fully connected layers and consists of two distinct almost symmetrical sub-networks (one for the critic and one for the actor) containing three layers each, plus one extra layer called ”log_std”, this extra layer has the shape of a vector with dimension equal to the action space, each value contained in this vector is used to create a normal distribution around the values of the actions taken by the agent, these distributions are used only in the training phase to take sub-optimal actions in order to explore the action space (the layer is not constant, it is trained as all the rest of the network). The two sub-networks are characterized by three connected sectors called: observation (zero layers), feature extractor (two layers) and network architecture (one layer), the observation sector isn't a part of the structure of the network, it represents the values of the state of the robot that have to be passed to the feature extractor. The only structural difference between the two sub-networks is the input of the network architecture, in the actor part is equal to the number of joints while in the critic part is one. All the agents in the MuJoCo environments created using PPO have this structure with seven layers, the only differences between them are the dimensions of the observation part and the output of the network architecture part of the actor sub-network, both depends on the environment.

C. Source policies

The starting environment of the transfer learning is the hopper environment, it has a continuous action space and consists of a simple robot composed by four parts (torso, thigh, leg and foot), connected by three joints, whose purpose is to go forward as fast as possible without falling (see Fig. 1). As said, from this environment two agents have been taken, one trained in the standard way (called standard agent) and the other using the UDR (called UDR agent). During the training of the UDR agent at each episode the masses of three (thigh, leg and foot) of the four body parts changed randomly following a uniform distribution around the nominal true value, the boundaries of each distribution were set to be plus and minus half of the nominal value. The two agents have been trained for 250000 timesteps with the same parameters, all the characteristics of the training have been defined with trial and error on the standard agent, the parameters have been chosen in order to maximize the reward while the amount of timesteps has been decided balancing reward improvement and training time. The agents obtained from the two source agents are divided in: UDR transfer agents and standard transfer agents.

D. The environments

The two target environments, half cheetah and walker2d, were chosen because of their similarity to the hopper environment, all of them consist of a legged robot, with different shape and number of leg, whose purpose is to go forward as fast as possible. Although the robots have the same purpose and belong to the same robot family, their structure (shown in Fig. 1) led to different observation and action spaces. Both the target environments have an observation space with 17 dimensions and an action space with 6 dimensions, while the source environment has an observation space with 11 dimensions and an action space with 3. Looking at the Gym documentation [1], can be seen that the values in all the action spaces represent the torque applied between links and the observation spaces have the same structure: 5 dimensions are used to give all the information (x,y,z coordinates, angle and angular velocity) about one specific point in the robot, to know its position and orientation; the others corresponds to

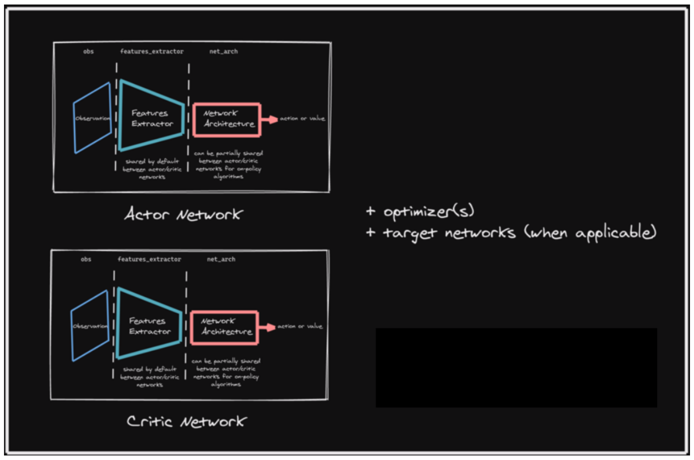
obs
features_extractor
net_arch
Observation
Features Extractor
Network Architecture
action or value
shared by default between actor/critic networks
can be partially shared between actor/critic networks for on-policy algorithms
Actor Network
obs
features_extractor
net_arch
Observation
Features Extractor
Network Architecture
action or value
shared by default between actor/critic networks
can be partially shared between actor/critic networks for on-policy algorithms
Critic Network
+ optimizer(s)
+ target networks (when applicable)

Fig. 2. High level representation of the structure of an agent created following the MLP model by the PPO algorithm, it is composed by two nearly symmetric sectors, one for the actor and one for the critic, they both have three parts: an observation space, a feature extractor and the network architecture.

the angles and the angular velocities of the joints controllable by the agent.

E. Policy transfer

Using the information in the documentation, the transfer was made by matching all the observations and actions in the source environment with the observations and actions in the target environment, then, all the parameters that refer to a specific observation (or action) in the source network were copied in all the parameters that refers to the matched observation (or action) in the target network, e.g., considering that the parameter of a fully connected layer can be seen as a matrix of dimension *out_channels*×*in_channels*, where *in_channels* is the number of observations, the parameters that refer to the angle and the angular velocity of the joint between the foot and the leg of the hopper, columns with indexes 4 and 10, were copied into the parameters of the angle and angular velocity of the lowest joint of one of the legs of the target agent, columns 4 and 13 for the rear (or right) leg or columns 7 and 16 for the front (or left) leg in the half cheetah (walker2d), and so on for all the observations in the source environment, the parameters of the observations in the target environment that didn't have a match were left to the initialization given by the algorithm. This particular procedure was done only for the first layer of the feature extractor part of both the sub-networks. For the layer in the network architecture part of the actor a similar procedure was followed, in this case, the copy was done by rows because *out_channels* is the number of actions and vary between the agents, all the other layers and biases were copied in their entirety.

Since the number of legs is different between the source and target environments and the shapes and purposes of the legs vary, the transfer has been made on different areas, on the two legs independently and on the two legs simultaneously, following the method explained before, to cover all the possibilities and obtain a wider amount of data, the third case is the union of the first two, the parameters of the observations (and actions) relative to the joints of the hopper were copied on both the two legs of the target environment. For simplicity the six transfers are called: rear transfer, front transfer, half cheetah full transfer, right transfer, left transfer and walker full transfer, the target of the first three is the half cheetah while the target of the remaining three is the walker2d. The only peculiarity among all the transfers is that in the half cheetah full transfer, the parameters of the rear leg in the network architecture part of the actor sub-network were copied changing their sign, this improved a lot the performances of the agents.

To analyze the effects of each part of the agent network, several transfers were made where only some parts of the source agent were transferred, the parts are:

- actor sub-network;
- actor sub-network and log_std layer;
- actor and critic sub-networks without the log_std layer;
- all the network.

In this way, a total of 24 agents were created for each destination environments (the four different source parts multiplied by the three destination areas multiplied by the two types of source agents). All of the obtained agents were trained for 0, 125000, 250000 and 500000 timesteps to analyze the adaptability and the resilience of the transfer.

IV. Experiments

A. Hopper

After the training, the standard and the UDR policies were tested on two different hopper domains that differ only on the weight of one part (the torso) by one unit, they are called source and test domain. The standard agent was trained on the source domain. The results of the tests are shown in Table I, for comparison, the table shows also the result obtained with the test domain by an agent (called test agent) trained on it. The standard agent performed very well with the source domain but poorly with the test domain, its policy is too much attached to its domain that it couldn't overcome a small change, instead, the UDR agent performs better than the standard agent on a unknown domain and, due to the variety in its training and the simplicity of the task, the UDR agent obtains slightly greater results of both the standard and the test agents on their own training domains.

B. Walker2d

The results of the experiments in the walker2d environment are shown in Table II. There are also reported the results of an agent built in the standard way and trained in this environment (base agent), this agent was used to tune the parameters to not favor a specific agent.

Without any tuning the UDR transfer agents obtained very good results but only one of them surpassed the standard transfer agents. In the actor transfer the UDR transfer agents performed better than the standard transfer ones in most of the cases, only in the left transfer and in the 500000 timesteps full transfer the results changed and the standard transfer agents performed better. Adding the log_std layer at the transfer changed the behaviour of the left and right transfers, in fact, in the left transfer the UDR agent surpassed the standard one two times out of three, while in the right transfer was the opposite. Overall, with this type of transfer, all the results obtained after 125000 timesteps of training were slightly improved but the others were worse, much worse in some cases, the confidence developed in the source environment, represented by the layer, didn't help much in this case. In the third type of transfer can be seen an improvement in more than half of the cases with

TABLE I
HOPPER AGENTS PERFORMANCES

Domains	Standard agent	UDR agent	Test agent
Source	1604.68	1658.21	-
Test	899.85	1082.90	1056.05

Mean rewards over 50 test episodes. Source masses: 2.53, 3.93 2.71, 5.09. Test masses: 3.53, 3.93 2.71, 5.09.
Parameters used: gamma=0.99, n_steps=2048, n_epochs=10, seed=5, learning rate=0.0015, n_steps is the number of step to run for each environment (in case of parallel training) per update.

TABLE II
WALKER2D RESULTS

n timesteps	Base agent	Right leg		Both legs		Left leg		Mean	
		standard	**UDR**	**standard**	**UDR**	**standard**	**UDR**	**standard**	**UDR**
Transfer of the actor sub-network									
0	163.81	145.56	106.73	138.19	494.30	494.27	453.29	259.34	351.44
125e3	343.92	403.65	1030.96	151.47	431.02	404.44	404.10	319.85	622.02
250e3	572.07	643.34	648.91	439.11	1061.72	1069.36	932.67	717.27	881.10
500e3	2576.213	720.95	1357.16	1188.20	1154.87	1327.79	1130.56	1078.98	1214.20
Transfer of the actor sub-network and the log_std layer									
0	163.81	145.56	106.73	138.19	494.30	494.27	453.29	259.34	351.44
125e3	343.92	638.60	1036.72	374.64	480.26	378.55	489.86	463.93	668.94
250e3	572.07	675.63	656.73	233.23	654.90	801.24	998.00	570.03	769.88
500e3	2576.21	826.29	610.30	490.61	953.05	1315.05	857.40	877.32	806.92
Transfer of the actor and the critic sub-networks									
0	163.81	145.56	106.73	138.19	494.30	494.27	453.29	259.34	351.44
125e3	343.92	374.90	371.93	407.04	595.54	361.52	487.32	381.16	484.93
250e3	572.069	1312.51	530.27	969.27	781.74	414.35	1061.47	898.71	791.16
500e3	2576.21	1986.71	2164.65	498.14	1266.67	905.88	1321.74	1130.25	1584.35
Transfer of all the network									
0	163.81	145.56	106.73	138.19	494.30	494.27	453.29	259.34	351.44
125e3	343.92	891.45	364.92	153.62	1308.96	343.70	427.27	462.92	700.38
250e3	572.07	1233.87	434.44	385.311	1106.73	455.15	534.12	691.44	691.76
500e3	2576.213	1642.55	928.46	367.4	957.83	950.288	1358.594	986.75	1081.63

The values are the mean rewards over 50 test episodes. The 'mean' columns represents the mean of the results in the same row.
The parameters used to train all the agents in this environment are: gamma = 0.99, n steps = 1024, n epochs = 10, seed = 3, learning rate = 0.0002 and batch size = 32, they are tuned to obtain the maximum results with the base model over 500000 timesteps.

respect to the first one, the UDR left transfer agent and the standard right transfer one benefited the most from this type of transfer. The full network transfer has the most defined behaviour, in the right transfer the standard transfer agent performed always better but in the other two cases the UDR won, in most of the cases the results were worse than the actor sub-network transfer.

Looking at the overall performance, the UDR transfer agents show more consistency, two-third of the times they obtained better results and the mean of their results is almost always greater than the standard transfer agents. The only case where the UDR transfer agents performed worse was in the right transfer, this behaviour is due to the robot itself and the behaviour of the agent, the robot is extremely similar to the hopper one, the right leg is exactly the same as the leg in the hopper, the other leg, despite being the main difference between the two environments, doesn't change much the behaviour of the agent, in fact, looking at the rendering, it can be seen that the left leg is only used to balance the robot, it is not used in moving forward. These aspects makes the walker environment almost equal to the hopper environment with the source domain, where the standard agent is more prepared, and the right transfer accentuates this similarity more than the other. The best result among all was obtained with a UDR transfer but the second, the third and the fourth best results were achieved with a standard transfer, anyway, neither of them surpassed the greatest result of the base agent.

C. Half cheetah

The results of the experiments in the half cheetah environment are shown in Table III. As before, there are also reported the results of an agent built and trained in this environment (base agent) used to tune the parameters. Without any training the UDR transfer agents seem to perform better, actually all the agents went backward, standard transfer agents obtained worse results because they covered a longer distance, with a more precise transfer they should be able to improve their results enough to surpass the UDR transfer agents but, because the focus is to compare the performances of different type of transfer and not to obtain the best one, no optimization was made. In the first type of transfer there wasn't a well defined behaviour, the standard transfer agents performed better than the UDR ones in a few more cases, overall the performances are almost always worse compared to the base model. In the actor and log_std transfer there is a pattern, in the rear and full transfers the UDR transfer agents performed better in almost all the cases while in the front transfer they obtained worse results, however, almost all the results were lower than the actor transfer, like in the other environment, the confidence represented by the log_std layer reduced the learning capacity of all the agents. The third type of transfer increased most of the performances of the standard transfer agents but reduced almost all the UDR transfer agents results, anyway, all the results are worse than the actor transfer. In the full transfer the behaviours were more defined, in the front and most of the rear transfer the standard agents achieved better results but in the others were worse, overall the results were the worst among all.

The UDR transfer agents were less consistent than the ones in the walker2d environment but the mean of their results was almost always greater than the standard ones, they managed to adapt slightly better then them to the different structure of the legs. In this environment, the more was transferred, the worse were the performances, in fact, the best results between the

TABLE III
HALF CHEETAH RESULTS

n timesteps	Base agent	Rear leg		Both legs		front leg		mean	
		standard	UDR	standard	UDR	standard	UDR	standard	UDR
Transfer of the actor sub-network									
0	-0.16	-294.87	-54.21	-268.82	-298.72	-486.61	-88.81	-350.10	-147.25
125e3	838.24	196.78	333.37	351.00	578.31	630.50	523.58	392.76	478.42
250e3	1223.49	282.47	799.32	737.26	623.12	876.83	862.17	632.19	761.54
500e3	1445.14	3397.01	1959.94	1197.27	897.33	928.05	1012.31	1840.78	1289.86
Transfer of the actor sub-network and the log_std layer									
0	-0.16	-294.87	-54.206	-268.82	-298.72	-486.61	-88.81	-350.10	-147.25
125e3	838.24	190.87	195.32	-201.93	131.36	275.80	158.01	88.24	161.56
250e3	1223.49	175.61	1003.87	-66.78	631.17	808.55	513.53	305.80	716.19
500e3	1445.14	699.35	1921.79	729.46	735.99	1205.31	480.41	878.04	1046.06
Transfer of the actor and the critic sub-networks									
0	-0.16	-294.87	-54.21	-268.82	-298.72	-486.61	-88.81	-350.10	-147.25
125e3	838.24	-109.48	382.39	187.60	-0.18	276.33	84.11	118.15	155.44
250e3	1223.49	448.48	390.82	567.92	443.84	851.31	703.37	622.57	512.68
500e3	1445.14	822.22	912.06	966.43	751.79	870.86	1039.90	886.50	901.25
Transfer of all the network									
0	-0.16	-294.87	-54.21	-268.82	-298.72	-486.61	-88.81	-350.10	-147.25
125e3	838.24	83.59	-148.49	-249.69	-46.41	409.83	-64.99	81.24	-86.63
250e3	1223.49	409.13	35.56	-99.74	490.32	696.45	274.38	334.28	266.76
500e3	1445.14	580.01	646.33	435.04	504.31	969.80	655.73	661.62	602.12

The values are the mean rewards over 50 test episodes. The 'mean' columns represents the mean of the results in the same row.
The parameters used to train all the agents in this environment are: gamma = 0.99, n steps = 1024, n epochs = 10, seed 3, learning rate = 0.0003, batch size = 64, they are tuned to obtain the maximum results with the base model over 500000 timesteps.

four transfer types were achieved transferring only the actor part. Contrary to the walker2d experiments, the highest reward was obtained with a standard transfer agent and the next two with UDR transfer agents. This time the best three results surpassed the base agent one, the initialization of those agents helped to find the correct behaviour which wasn't learnt by the base agent, the tests showed that the base agent first capsized and then went on while the other went straight on with a smooth run.

V. CONCLUSIONS

The domain randomization is a training method born to overcome the sim-to-real transfer problem in the reinforcement learning field, its purpose is to build a policy enough robust to surpass small inaccuracies in the simulated environment with respect to the real one, the downsides of this method are the performance, which will probably decrease (not always, as shown in Table I), and the amount of information that the agent should unnecessary adapt to, if they are too many, the policy created would be not suitable for any environment.

This paper showed a possible application of this particular method in a field for which it is not thought, the across-embodiment transfer learning. The source agent was training with the uniform domain randomization applied to the masses of the body. The analysis were done making many target agents, with different training times and initialization, to obtain a wide amount of data to observe the performances obtained by the transfer in many conditions.

Reminding that any form of optimization or parameter tuning was done for any of the agents built with the transfer, the results show that the agents built starting from the UDR agent are more consistent, in particular, if only the actor part is transferred, these agents perform better than the standard transfer ones in most of the cases, if we transfer more, the behaviours are not well defined but the results are comparable if not higher in most of the cases. Furthermore, the transfer of the log_std layer reduces the performances of all the agents, in both the environments, while the transfer of the critic part can improve the results, depending on the similarity of the two environments, in the walker2d it improved the results, in the half cheetah it didn't.

Starting from these analysis, i can say that in the across-embodiment transfer learning an agent trained with uniform domain randomization is a safer source agent than one trained standardly, especially if only the policy is transferred. However, using a standard agent is a good choice if the finetuning needed is low, which means if the environments are very similar in many aspects (like weights, structure and reward).

This work is just the beginning, there are many possible analysis that must be done in many aspects in the transfer pipeline to understand the possible benefits of the domain randomization in this field, from the effects of different domain randomization to the results in increasingly different environments. However, the results of this analysis show that the domain randomization has the potential to be a useful tool also in the across-embodiment transfer learning

REFERENCES

[1] Gym documentation: https://www.gymlibrary.dev/environments/mujoco
[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). "Proximal policy optimization algorithms", PDF