Universal Notice Networks: Transferring Learned Skills Through a Broad Panel of Applications

Mehdi Mounsif, Sébastien Lengagne, Benoit Thuilot, Lounis Adouane

Universal Notice Networks : Transferring learned skills through a broad panel of applications.

Mehdi Mounsif[1], Sébastien Lengagne[1*], Benoit Thuilot[1] and Lounis Adouane[2]

[1]ISPR, Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Institut Pascal, Clermont-Ferrand, 63000, France.
[2]Heudiasyc, Université de technologie de Compiègne, 57 avenue de Landshut, Compiègne, 60203, France.

*Corresponding author(s). E-mail(s): sebastien.lengagne@uca.fr;
Contributing authors: mounsif.mehdi@gmail.com;
benoit.thuilot@uca.fr; lounis.adouane@hds.utc.fr;

Abstract

Despite great achievements of reinforcement learning based works, those methods are known for their poor sample efficiency. This particular drawback usually means training agents in simulated environments is the only viable option, regarding time constraints. Furthermore, reinforcement learning agents have a strong tendency to overfit on their environment, observing a drastic loss of performances at test time. As a result, tying the agent logic to its current body may very well make transfer unefficient. To tackle that issue, we propose the Universal Notice Network (UNN) method to enforce separation of the neural network layers holding information to solve the task from those related to robot properties, hence enabling easier transfer of knowledge between entities. We demonstrate the efficiency of this method on a broad panel of applications, we consider different kinds of robots, with different morphological structures performing kinematic, dynamic single and multi-robot tasks. We prove that our method produces zero shot (without additionnal learning) transfers that may produce better performances than state-of-the art approaches and show that a fast tuning enhances those performances.

Keywords: Control, Transfer Learning, Cooperative Control, Reinforcement Learning

1 Introduction

In recent years, Reinforcement Learning (RL) methods have been successfully applied to a broad range of complex, high-dimensional and non-linear problems, both in simulated environments [1–3], and in the real world [4–7]. While an important part of these works relies on model-free techniques, an increasing focus is set on model-based methods, given their higher sample efficiency and their ability to plan a successful sequence of actions [8], [9]. However, in both model-based and model-free settings, the final behaviour is critically sensitive to the training distribution. As a result, the system performances decrease when placed in an environment with a test distribution too far from the training distribution, which is the case when a model trained on simulated environments is deployed to the real world. Moreover, even when the resulting policy is able to cope up with the intrinsic differences between virtual and physical world, it is usually not possible to transfer this skill to another agent with a different body configuration. Commonly, transfer implies replacing the last layer of the network to fit the new configuration and task [10]. This modification may produce performance loss when no precaution is taken, because the learning process will not isolate control parameters, depending on the robot properties, with decisional/logic units that are related to the task, hence changing a complete layer may remove critical weights, effectively plummering the agent's efficiency at its task.

Indeed, deep learning techniques uses backpropagation and gradient descent to optimize the parameters of a neural network based on a loss function. Training is usually done with randomly sampled batches from the dataset and the parameters of the network are usually initialized with random weights. Using certain weights distributions such as [11], improves learning speed and final performances. But overall, it is difficult to predict the parameters trajectory in their space during training session. In the reinforcement learning context, this stochasticity is emphasized given the reward signal dependency to the agent actions as well, which are themselves sampled from a distribution depending on the parameters. This inherent complexity does not necessarily prevent the agent from learning, however, there is no principled method to predict which part of the neural network is responsible for specific behaviours, at least for straightforward architectures.

The Universal Notice Network (UNN) approach [12] proved on various 2D tasks that it is possible to tackle specifically this drawback, by ensuring knowledge is held in predetermined areas of the model. Relying on this approach permits to segment learning between various modules (one module for the task and two modules for the robot control), thus knowledge to solve a task is isolated from control logic. It is then possible to transmit skills to solve this specific task in a plug-and-play fashion between robots with different configurations. While these configurations can be physically defined, such as various numbers of joints, segment lengths and so on, it is also possible to think of them as the differences between the simulated-world robot and the real-world robot,

or, in a more pragmatic view between a robot and its damaged or time-altered version.

If these robots are tasked with the same assignment and if skills for solving the task are distinct from those needed to control the robot, then one could simply transfer the trained task module to get the robot to perform, hence avoiding repeated training. To sum up, the main contributions of this paper are the following:

- an application of the UNN method [12] to 3D environments for manipulation and cooperative tasks,
- the introduction of a new framework: the Heuristic Curriculum Learning (HCL) that aims at minimizing the correlation between the way the agent succeeds in the task and its body configuration,
- a practical and specific example demonstrating how the bias inducted by modules pretraining can positively improve the learning process of a UNN.

Those contributions are evaluated on 3D simulated environments that are presented in the following video : https://youtu.be/bhxOSiZjANo.

2 The Universal Notice Network Method

2.1 UNN pipeline

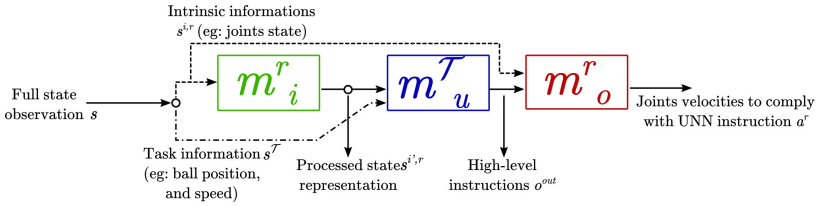


Fig. 1: UNN pipeline considering the three sub-modules $m^r_i, m^{\mathcal{T}}_u, m^r_o$ and the observation state decomposition.

In practice, the UNN pipeline is an end-to-end model, composed of three sub-modules, $m^r_i, m^{\mathcal{T}}_u, m^r_o$, as shown in Figure 1. The first and the last part of this model m^r_i, m^r_o, respectively the input and the output modules, are specific to the robot r, while the center module $m^{\mathcal{T}}_u$, the UNN, is designed to be robot-agnostic and specific to the task $\mathcal{T}$. Hence, given a task $\mathcal{T}$, we can observe at each timestep the full state observation vector s. We split this vector into two parts: $s^{i,r}$ holding data intrinsic to the considered robot and $s^{\mathcal{T}}$ task-related information, independent from the agent. The first step is to feed the intrinsic information $s^{i,r}$ to the input module m^r_i that produces a processed states representation $s^{i',r}$:

$$s^{i',r} = m^r_i(s^{i,r}) \tag{1}$$

Then, merging this processed states representation $s^{i',r}$ and the task-related information $s^{\mathcal{T}}$ we get the input of the UNN that will output a high-level instruction

$$o^{out} = m_u^{\mathcal{T}}(s^{\mathcal{T}}, s^{i',r}) \tag{2}$$

The effective robot action a^r is then recovered by concatenating the initial intrinsic vector $s^{i,r}$ with o^{out} and feeding it to the last module, that is:

$$a^r = m_o^r(o^{out}, s^{i,r}) \tag{3}$$

Finally, the whole process can be synthetized as:

$$a^r = m_o^r(m_u^{\mathcal{T}}(s^{\mathcal{T}}, m_i^r(s^{i,r})), s^{i,r}) \tag{4}$$

In practice, for any module, that is, either input and/or output module or even the UNN, there is no theoretical restriction on the computational model used. As long as the function used returns a vector of the expected size, functions approximators as well as analytical methods, if available, are perfectly suitable.

One can see the robot modules m_i^r, m_o^r as a model that allows the agent to achieve basic movements. On the one hand, as an example, if we consider tasks involving robotic manipulators, the input module m_i^r could compute the effector configuration based on the robot's joints information and the output module m_o^r could convert the effector desired configuration to angular velocities for each articulation. While, on the other hand, the UNN module $m_u^{\mathcal{T}}$ is a high level model that produces the desired effector configuration to perform the task from the current effector position.

Ultimately, for any UNN/agent couple, the goal is, to find the module functions $m_i^r, m_u^{\mathcal{T}}, m_o^r$, that produce the movement $\mathcal{M}$ generated by actions a^r provided by the pipeline for solving a task $\mathcal{T}$ and that must ensure the set of constraints $g_{\mathcal{T}}$ to perform the task and the set of constraints g_r to ensure the physical limits of the robot, such as:

$$\begin{aligned} &\text{find} && m_i^r, m_u^{\mathcal{T}}, m_o^r \\ &\text{such as} && g_\tau(\mathcal{M}) \leq 0 \\ & && g_r(\mathcal{M}) \leq 0 \\ &\text{With:} && \mathcal{M} = f(a_r) = f(m_o^r(m_u^{\mathcal{T}}(s^{\mathcal{T}}, m_i^r(s^{i,r})), s^{i,r})) \end{aligned} \tag{5}$$

The following section proposes a framework to solve problem 5 by sequentially computing the robot and the task modules.

2.2 Building and using UNN modules

The motivation behind the UNN approach is to ease and enhance the efficiency of skills transfer between agents of different configurations. In Figure 2, we define a set of several robots with different configurations and a set of tasks achievable by these robots. The goal of the UNN is that any robot of the set can perform task. We provide here a possible workflow description relying on

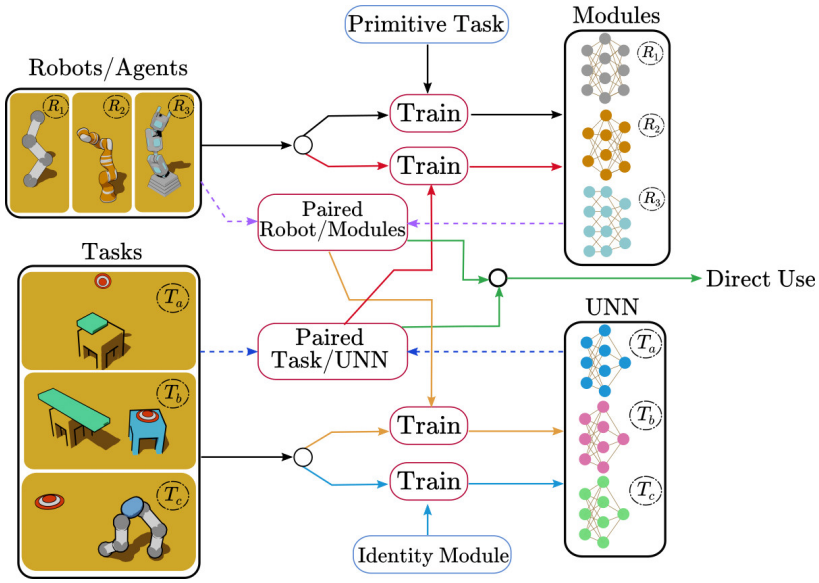


Fig. 2: The UNN workflow for creating and using modules and UNNs. Dashed violet paths are to pair the robot with its module. Dashed blue paths are to pair the task with its UNN. Black and red paths are to create new robot modules. Orange and blue paths are to create new task module (UNN).

the UNN, and summarized in Figure 2 to create new robot modules, new UNN (task) modules and to use it.

2.2.1 Use of the existing modules

Now, consider the case where a user wants a robot r to perform a given task $\mathcal{T}$ and assume that the robot modules and the UNN module already exist. It opens the possibility for a user to pair any robot up with its module with a UNN relevant to the current task and carry on with the realization (green path on Figure 2).

2.2.2 Creating robot modules

If no robot modules are available, it is possible to generate them, using a primitive task: this is the robot module creation (black path in Figure 2). Alternatively, a robot module can be created using a given task and its paired pretrained UNN (red path).

2.2.3 Creating UNN (task) modules

If the robot modules are available, the next step is to assess whether a trained UNN module is available for this specific task. If not, the user should create it using one out of two ways: relying on its trained robot modules to train the

UNN (orange path) or using Identity modules such as the HCL framework (blue path), explained in Section 2.3.

2.3 Heuristic Curriculum Learning

However, due to the notable RL tendency to fall into local minima, the UNN is likely to discover a successful strategy for solving the task that depends on its body configuration (for instance, blocking an object between two articulations). In these deceptive cases, the constraint related to physical limits in Equation(5) may no longer be respected. While this is not an issue for the current agent, it is detrimental for the efficiency of a future transfer to an agent with a different structure. To ensure that the UNN constraints are not entangled with the agent ones, the Heuristic Curriculum Learning (HCL) is introduced. This approach uses the same algorithms as the baseline UNN method, but relies on an environment modification that assimilates the robot to its effectors, thus creating a virtual robot, as shown in Figure 3. On the formulation side, we set the base models m_i^r, m_o^r as identity functions and provide directly the UNN output to the command. These two modifications release most of the constraints that would have been distilled by the pre-trained base and prevent the UNN from learning a configuration-specific strategy, resulting in the creation of an UNN closer to a model-agnostic setting.

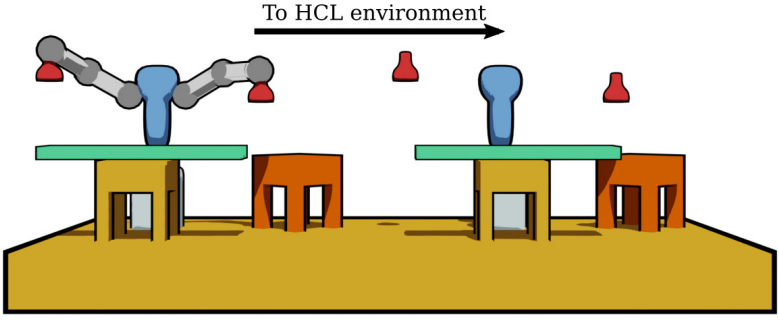
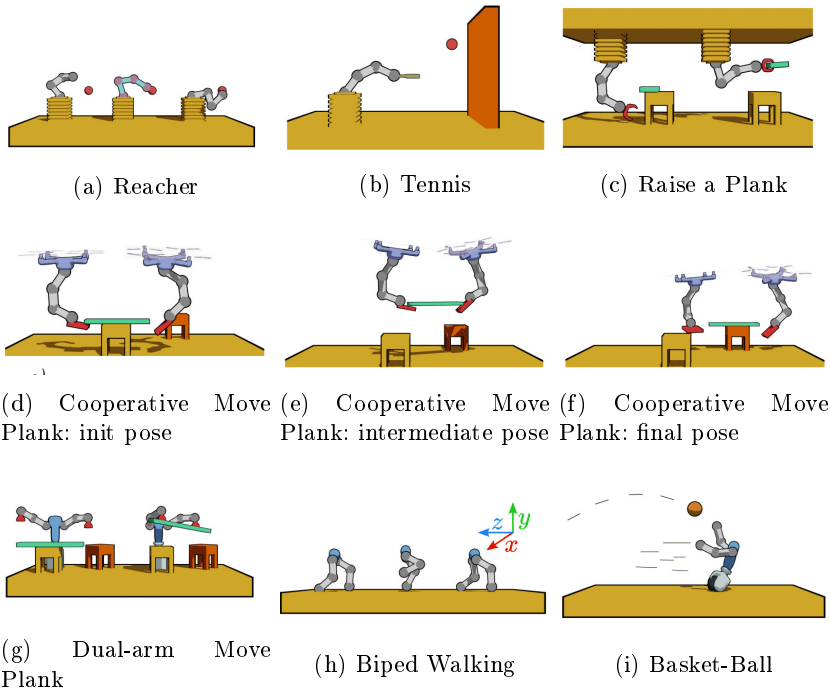


Fig. 3: An HCL environment is created by removing the kinematic constraints inherent to the agent's body and assigning an identity function instead to translate directly the HCL outputs into actions



(a) Reacher (b) Tennis (c) Raise a Plank

(d) Cooperative Move Plank: init pose (e) Cooperative Move Plank: intermediate pose (f) Cooperative Move Plank: final pose

(g) Dual-arm Move Plank (h) Biped Walking (i) Basket-Ball

Fig. 4: A panel of the tasks used to evaluated the performance of the UNN.

3 Experimental Setup

In this section, we introduce the seven environments, as presented in Figure 4, that we implemented using ML-Agents framework [13]. As we train the UNN using reinforcement learning (RL), we detail the general concept and the Markov Decision Process (MDP) reward function that motivates the agent behaviour. In those environments, the UNN module receives the same information $s^{i'}$ from its input module and outputs the same information o^{out}. Specifically:

- The processed intrisinc information $s^{i'} \in \mathbb{R}^9$ consist in the end effector linear position and velocity and orientation.
- The UNN output $o^{out} \in \mathbb{R}^6$ is composed of desired effector linear velocity and angular velocity.

For environments a to c, i.e. environments involving a single arm, the general pipeline is the one depicted in Figure 1. This approach is adapted as shown in Figure 5 for tasks d to g, given that multiple arms are controlled. For the latter, the main difference is that the UNN will receive the processed intrisinc information for each arm and output o^{out} for each effector.

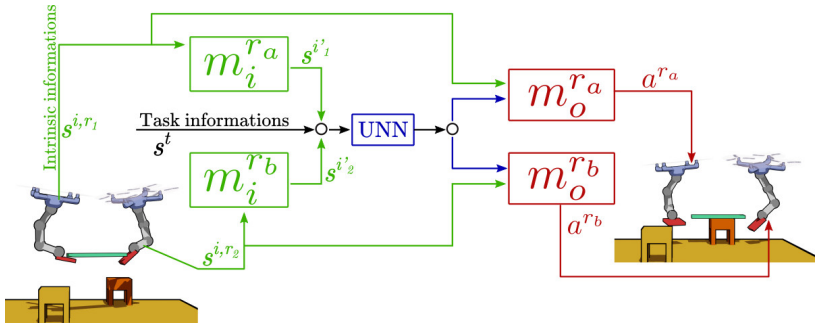


Fig. 5: Architecture for cooperative tasks.

3.1 Robots

The UNN method goal is to create a module encompassing knowledge independently from the agent accomplishing the task in order to later have the possibility to transfer this knowledge module to an agent with a different configuration. In this view, Figure 6 shows the robots used to embody this transfer, along with their Degrees of Freedom (DoF) (drawn as colored arrows). Some architectures are inspired from real-world robots, such as the Kuka-LWR (5 DoF) or Berkeley Blue (5 DoF). We also introduce the Generic-3 robots, a 3 joints with 2 DoF for each joint configuration, and an architecture declined into two variants (Leg Type 1, 3 DoF, and Leg Type 2, 4 DoF) that were used in the Biped environment.

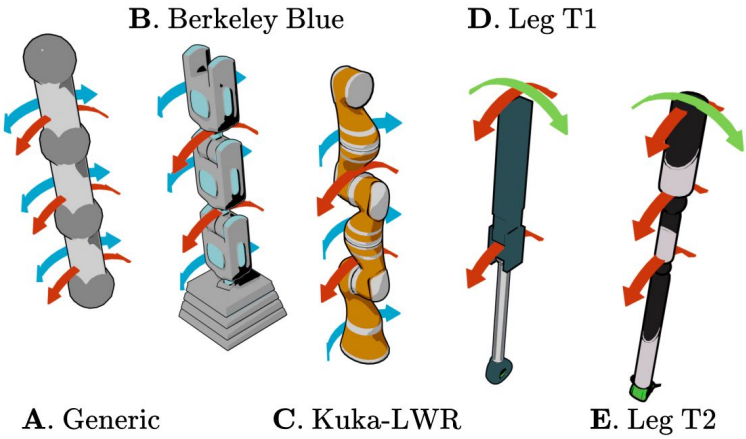


Fig. 6: From left to right: Generic robot, Berkeley Blue, Kuka-LWR, Leg Type 1, Leg Type 2

3.2 Reacher

The reaching task is presented in Figure 4*a* and consists in setting the position of the end-effector to a desired position. This is a fundamental and particular piece in our framework, since most of the experiments will require the agent to have mastered this skill. That task is considered as the primitive task introduced in Figure 2. However, while seemingly very basic, an agent trained with a reward depending solely on the distance between its effector and the target very rarely yields a policy with plausible/desirable behaviour. Most of the time, that kind of reward function ends up into a policy with unexpected behaviour at best and unstable at worse (Figure 4*a*, right-most robot). Hence, to avoid this drawback of reinforcement learning, we take advantages of various insights from [3] and we propose the following MDP for learning:

- $s^{\mathcal{T}} \in \mathbb{R}^7$: Vector from the end-effector to the target and target orientation (as quaternion), expressed in world frame
- Reward function R_t: Inversely proportional to the distance between current agent configuration q_a (Figure 4*a*, left-most robot) and a configuration given by a state-of-the-art method q_r, here [14] (Figure 4*a*, central robot). Thus:

$$R_t = \alpha \times \|q_a - q_r\|^{-1} \tag{6}$$

  where α is a scalar used for reward normalization. The q_r configuration is the main leverage used to induce bias within output module. Indeed, in this specific task, the UNN is transparent, that is acting as the identity function. Thus, given that training focuses on the output module, by defining wisely the q_r configurations in this task, it is possible to generate output modules that, for instance, can conditionally behave in a way that can recall inverse kinematic models constrained through pole targets. This feature is particularly useful and its usages are more thoroughly detailed in Section 4.

3.3 Tennis

This task consists in a self-play in a tennis-like setting, see Figure 4*b*. A paddle effector is considered as the end-effector of a robotic arm and the goal is to maximize the number of times the ball bounces against the wall and the episode ends up whenever the ball drops below a preset height threshold. Thus, the MDP has the following components:

- Task-related state $s^{\mathcal{T}} \in \mathbb{R}^6$: ball linear position and velocity, expressed in world frame
- Reward function:

$$R_t = \begin{cases} 1 & \text{if contact} \\ 0 & \text{else} \end{cases} \tag{7}$$

  For each episode, the reward is incremented for each bounce, until the ball drops or if the time limit is reached.

3.4 Raise a Plank

This manipulation task consists in reaching and raising a plank to a constant desired height h_d, see Figure 4*c*. The environment is composed of a fixed support for the plank, the plank itself and a robotic arm equipped of a gripper. For this environment, we have:

- Task-related state $s^{\mathcal{T}} \in \mathbb{R}^9 \times \mathbb{B}^1$: plank linear velocity and angular velocity, expressed in world frame, plank position expressed in end-effector frame, and the contact flag c between the gripper and the plank,
- Reward function:

$$R_t = -(1-c) \times d + c \times \exp(- \mid h - h_d \mid) \tag{8}$$

  With d the distance between the effector and the plank and h the actual plank height.

3.5 Cooperative Move Plank

The cooperative Move Plank environment tackles a task requiring coordination to move a large object, beyond the capabilities of a single robot. In practice, this environment features two robotic arms and a large plank to be moved from its support to another location (see Figure 4*d*). Two target points are defined on the plank for each end-effector. The UNN uses the process described in Figure 5. Formally, we have:

- Task-related state $s^{\mathcal{T}} \in \mathbb{R}^{16} \times \mathbb{B}^2$: plank linear position in each arm end-effector frame, plank linear position expressed in goal frame, plank orientation and angular velocity, expressed in world frame and c_i, the contact value between effector i and the plank.
- Reward:

$$R_t = \begin{cases} -\alpha \times (D_1 + D_2) & \text{if } \sum_{i=1}^{2} c_i = 0 \\ 0 & \text{if } \sum_{i=1}^{2} c_i = 1 \\ \frac{\beta}{D_t} & \sum_{i=1}^{2} c_i = 2 \end{cases} \tag{9}$$

  with α, β two positive scalars used for reward normalization given the simulation scale, D_i the distance between the effector i and its target points. Finally, D_t is the distance between the plank and the goal position.

A variant of this environment is the Dual-Arm setting, see Figure 4*e*. While similar in concept, it presents the following differences: both arms belong to a single body and each arm is equipped with a suction pad. Consequently, we enhance the state $s^{\mathcal{T}}$ with a contact flag for each suction pad.

3.6 Biped walking

To broaden the range of applications using the UNN method, we introduce the walking biped task. In this environment, we want a biped robot, composed of

a pelvis and two legs, to walk across a platform, see Figure 4*f*. We assimilate this application to the centralized control of two reacher robots and, similarly to the comanipulation task, the pipeline consists in a single UNN setting target positions for each leg, as shown in Figure 5. The MDP for the walk UNN described below requires a slightly more complex reward function:

- $s^{\mathcal{T}} \in \mathbb{R}^{11}$: Pelvis linear position in the center of masses (CoM) frame, pelvis velocity and angular velocity in world frame, pelvis height from the ground, pelvis lateral offset from the middle of the platform.
- The reward R_t is composed of various terms, that is:

$$R_t = R_f + R_l + R_h \tag{10}$$

with:

  - the pelvis speed along the forward axis:

$$R_f = \max(V_f, 0) \tag{11}$$

    where V_f is the pelvis speed along the forward axis,
  - the lateral offset between the pelvis and the forward axis, to encourage the agent to walk in a straight line:

$$R_l = \exp(-|P_l|) \tag{12}$$

    where P_l is the lateral pelvis position regarding the forward axis,
  - the vertical offset between the pelvis and the target height (initial pelvis height, in practice), to prevent the agent from jumping:

$$R_h = \exp(-\left|P_v - P_{v,0}\right|) \tag{13}$$

    where $P_v, P_{v,0}$ are respectively the vertical pelvis position and the target height position.

3.7 Basket-Ball task: catching the ball

The Basket-Ball environment, see Figure 4*g*, aims at testing a dual arm setup with a holonomic mobile base. The agent must catch a ball and keep it at a constant position until the end of the episode. When the environment resets, the ball is thrown with an initial velocity. We define this environment as the following MDP:

- $s^{\mathcal{T}} \in \mathbb{R}^{6}$: Ball linear position and velocity in world frame;
- o^{out}: in addition to the usual output, the UNN also returns the forward and lateral velocity for the mobile base,
- The reward is defined as:

$$R_t = \alpha + \exp\left(-V_{ball}\right) \tag{14}$$

where V_{ball} is the ball speed, α a positive scalar ensuring the agent finds an initial solution. If the ball drops below a preset height threshold, the reward is zero and the episode resets, hence putting a stop to the agent reward accumulation.

4 Results

In order to demonstrate the versatility of our method, we show that it is possible to repurpose a robotic arm for various tasks and that the knowledge gained using this configuration can be used with very little to no retraining by other agents configurations. Our agents are trained using Proximal Policy Optimization (PPO) [15], an on-policy, actor-critic with trust-region deep reinforcement learning algorithm, which goal is to maximize the cumulative reward per episode. Using PPO implies that every module trained in this fashion will feature an actor model as well as a critic function. As for the network architectures, we rely on hyperparameters featured in other similar works such as in [3, 15, 16], although other configurations are likely to yield results as well. Specifically, both the UNN policy and the output robot module network have two hidden layers with respectively 128 and 64 units, with TanH activation function. Our reference baseline, the *vanilla* PPO policy has three hidden layers of 128 units, again with TanH non-linearity. Given that all our robots have a serial structure, we propose to affect a Forward Kinematic analytical model to the input robot module to compute $s^{i',r}$, such that $s^{i',r} = m_i^r(s^{i,r}) = \mathrm{FK}(s^{i,r})$. To evaluate transfer efficiency, both regarding learning and final performances, we consider four configurations:

- **PPO**: A baseline *vanilla* PPO agent is trained
- **PPO Transfer**: A new agent is initialized with the previously learned weights from the *vanilla* agent. In this case, the hidden weights are fixed during training and the optimization process affects only the input and output layer weights.
- **UNN**: The UNN is trained from scratch while being paired with a trained robot module or in its HCL version
- **UNN Transfer**: The previously learned UNN is transmitted to another robot, using its own trained robot modules. When the training percentage in the various table results is superior to 0%, it means that the bases are fine-tuned for this percentage of the initial training time. Specifically, the UNN weights are fixed during retraining and only the module behaviour will be updated by training.

The inherent structural differences between a *vanilla* PPO agent and a UNN agent have resulted in a slightly superior number of parameters in favor of the UNN. While it is legitimate to consider that both agents should have the same structure to ensure fair comparison, setting up such a configuration proved rather challenging and incited us to instead strive for a network configuration in which both models have at least two hidden layers. In practice,

however, if small changes in hyperparameters could indeed slightly enhance or hurt performances, most of the experiments presented below show a non-negligible difference between the UNN method and PPO baselines, which is unlikely to be absorbed by a supplementary layer.

The graphs used in this paper to evaluate the agents performance evolution are a common RL tool. Specifically, they display the magnitude of the mean reward per episode on the y-axis over the duration of training, which is represented on the episodes on the x-axis. Given the RL mathematical formalisation and goal, it is easy to understand that curves that quickly increase are favored and usually imply that the agent is learning how to behave efficiently in the environment.

If we consider the UNN retraining pipeline, the adaptation of a robot and its modules to a newly paired UNN (involving modifying the weights of the robot modules) may seem counter-intuitive, since those modules are closely related to physics, which is usually not subject to change. Given the following applications (relatively straightforward solutions and simple robots), our hypothesis is that the robot modules are able to twist slightly their physical understanding of primitive movements to adapt to the UNN instructions for performing the task, as shown in the following sections. Nevertheless, we believe that for higher level tasks, more abstract modules, ie. not necessarily transmitting a physically-grounded set of values, may be needed to keep this pipeline.

In the next sections, we demonstrate that the UNN framework make transfert possible over a set of various tasks. We will focus on three representative tasks: Tennis, Dual-Arm Raise Plank and the Biped Walk (summary video available at:https://youtu.be/bhxOSiZjANo). For each of these tasks, we compare the transfer efficiency when using the UNN approach with the straighforward transfer and evaluate the final policies performances using metrics specific to each environment. Additional results for environments Raise Plank, Cooperative Move Plank and Basket-Ball are available in the annexes. Then, we discuss the influence of the module bias within these environments, highlighting cases where that bias can be beneficial and where the HCL framework proves most useful. Finally, we present two additional ways to adapt or retrain the modules by using a pre-trained UNN. The implementation used in this paper is available on https://github.com/MoMe36/UNN_UnityEnvAndScripts. This Section presents focuses on some results, more details are available in Appendix A and in [17].

4.1 Skill transfert: tennis task

Let us first consider the Tennis environment, for which learning curves can be seen in Figure 7. In this case, the original PPO policy and the UNN are trained for 1 million timesteps, controlling the Kuka robot. It clearly appears that the UNN policy (dark blue) is superior by an important margin to the *vanilla* PPO policy, both in terms of learning speed and final performance. We detail various mechanisms fueling this phenomenon further. Once training is

over for those two policies, their trained models are transferred to a new agent with a different body configuration, specifically the Berkeley Blue robot. In particular, for the UNN case, the trained UNN for the tennis task is inserted in the pipeline of another agent with its trained bases and for the *vanilla* case, it is the hidden layers that are transferred. The learning curves for the new policies with transferred core are drawn in orange and light blue, respectively for the PPO transfer and UNN transfer. Several points are worth noticing in this configuration. First, the UNN transfer case immediately starts with a higher mean reward than the policies trained from scratch. In this specific case, the initial UNN transfer mean reward is even higher than the final mean reward of the *vanilla* PPO policy. Then, the progress rate for the transferred policy is much steeper than the original policy and only 10% of the initial training time are needed to recover a similar final mean reward.

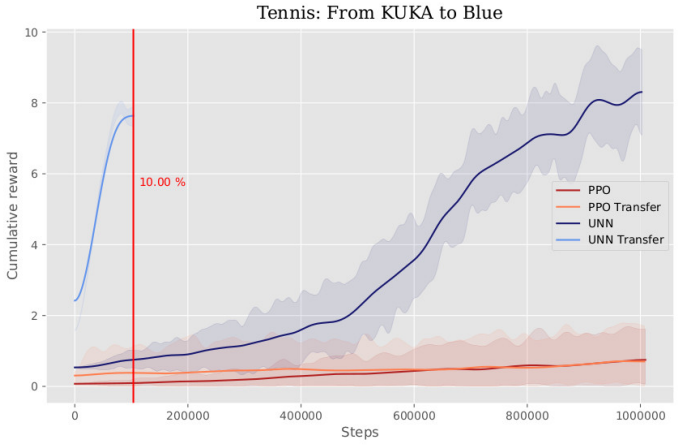


Fig. 7: Learning performances in the Tennis environment. PPO and UNN are performed using Kuka robot, transfer to Berkeley Blue robot

Table 1: Tennis Summary

		Config	**Training**	**Performances** (in bounces)
UNN	Original	Kuka	100%	18.74
	Transfer to	Berkeley Blue	0%	11.88
	Transfer to	Berkeley Blue	10%	17.56
	Transfer to	Generic 3	15%	18.12
PPO	Original	Kuka	100%	7.89
	Transfer to	Berkeley Blue	0%	0.86
	Transfer to	Berkeley Blue	100%	6.81
	Transfer to	Generic 3	100%	7.57

Once training is over, we evaluate the final policy performance by averaging the number of bounces against the wall for 100 episodes. These results are detailed in Table 1. This table and the subsequent tables all share the same organisation. First column gives the training case, whether the policy considered is the original or a transferred one. The second column, entitled configuration, details which robotic arm, or dual-arm, was used for this simulation. Third column precises the percentage of total training steps used for the performances of this policy, listed in the last column. In this case, these results confirm the performances predicted by the learning curves, as well as the transfer efficiency. In particular, we see that after training for 1 million timestep (100%), the UNN policy is able to bounce the ball over twice as much as the *vanilla* PPO policy. Also, we see that with no retraining, the UNN controlling the Berkeley Blue robot outperforms both the original PPO with the Kuka as well as the transferred PPO to the Berkeley Blue robot. In Table 1, a transfer toward the generic case has also been added, further emphasizing the domination of the UNN on this task.

The above results are prone to raise several questions about the reasons why UNN agents outperform *vanilla* PPO policies. The visual observation of each agent performance as well as the training context of its module reveal important qualitative facts. In this case, the output module is a network trained on a reacher task, using a reward function that encourages the agent to propose a solution that respects certain constraints, as described in 3.2. Hence, even with randomly initialized UNN module weights, the agent initially returns body configurations that are close to the ones needed to be able to easily bounce the ball back towards the wall. It is a good starting point and, through the restrictions it brings, helps the UNN policy find solutions in the current MDP. On the contrary, the *vanilla* PPO policy has no particular reason, when initialized, to propose those expected configurations and freely explores its action space. Thus, the policy may potentially receive its first positive rewards while being in a non-optimal position that will then see its probability increased thanks to the RL process. This is further exacerbated when the reward function is sparse and requires an important amount of tweaking to ensure the *vanilla* PPO policy receives its first rewards in a configuration that would allow it to keep improving.

4.2 The curious Biped case

Analyzing the curves from the Biped runs, see Figure 8, seems to yield partial contradictions to the precedent conclusions. Firstly, the *vanilla* PPO policy mean reward evolution along training clearly exceeds the UNN mean reward. And, while the transferred PPO policy does not clearly benefit from the transfer, apart from a steadier evolution, its mean reward grows also faster and reaches higher level than the UNN mean reward. Even though the transfer in the UNN case yields an important initial advantage for the new policy, the final mean reward stays way below both the *vanilla* PPO approaches. At test time, we compare the agent performances by measuring the average distance

(expressed in meters) covered by the considered agent over 100 episodes, see Table 2. In this case, interestingly enough, the UNN agent and its transferred version outperform both the PPO versions.

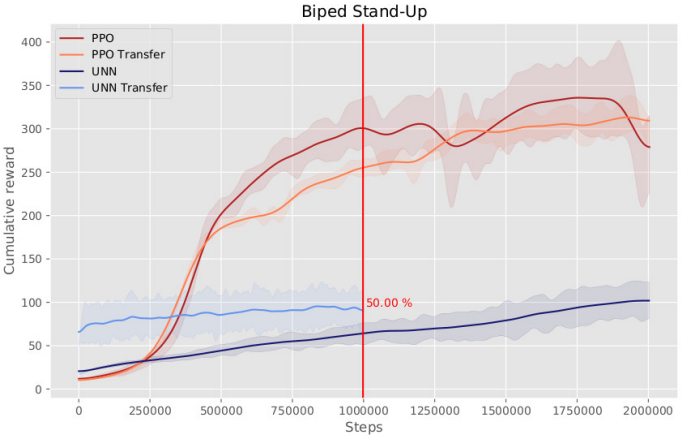


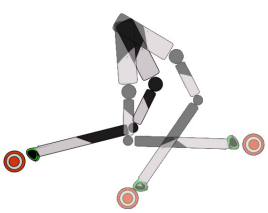
Fig. 8: Learning performances in Biped Walk environment. PPO and UNN are performed using 2 legs of Type1, transfer to robot composed of Leg Type1 and Leg Type 2.

Table 2: Biped Summary

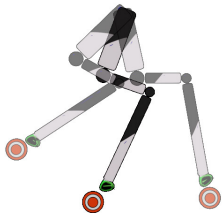
		Config	**Training**	**Performances** (in meters)
	Original	Leg Type 1/1	100%	49.74
	Transfer to	Leg Type 1/2	0%	21.02
UNN	Transfer to	Leg Type 1/2	30%	44.55
	Original	Leg Type 1/1	100%	29.63
	Transfer to	Leg Type 1/2	0%	2.90
PPO	Transfer to	Leg Type 1/2	100%	31.67

In this specific environment, the output module pretraining influence is even more pronounced. Indeed, in this case, the output module was trained again on a reaching task, but this time, we enforce through the reward function a bias for a knee back semi-folded leg configuration, as shown in Figure 9a and 9b. Hence, when paired with the UNN, the module interprets the actions ordered by the UNN through this particular bias which leads to a very different gait evolution (cf. Figure 9b) between the UNN policy and the *vanilla* PPO policy which freely accesses to all its action space (cf. Figure 9a). As a result, the trained *vanilla* PPO policy exhibits a non-symmetrical gait, in which one leg is always forward, while the other leg stays behind, see Figure 9c. This

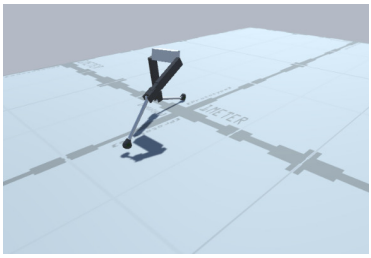
moving position allows the *vanilla* PPO policy to gather at least two reward terms out of three, namely the offset reward R_c and the height reward R_h, thus explaining the higher PPO score in Figure 8. However, as shown in Table 2, this approach does not optimize for the walking distance.



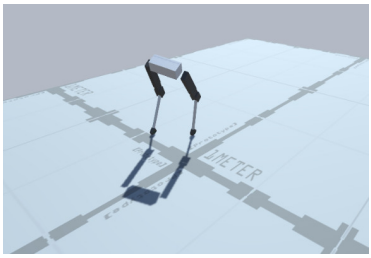
(a) Free configurations



(b) Enforced semi-folded configurations



(c) *vanilla* PPO gait



(d) UNN gait

Fig. 9: Influence of vanilla PPO and UNN approach on the leg configuration and on the walking gait.

Indeed, this policy is able to move slowly forward by giving quick impulses to its joints, in a stutter-like manner. On the contrary, the trained UNN policy is much more symmetrical and alternates between small jumps and walk to move forward, see Figure 9d, resulting in a larger distance covered, see Table 2.

4.3 A HCL configuration: Dual-Arm Raise Plank

In the Dual-Arm Raise Plank task, we compare the *vanilla* PPO approach with both the classic UNN and the HCL version. In this configuration, the HCL policy controls the heuristic version of a two-armed manipulator, as depicted on the right-hand side in Figure 3, while the classic UNN and the *vanilla* PPO policy are trained in an environment with two Kuka arms and transferred to two Berkeley Blue Robots. Learning progresses of the HCL, the classic UNN, *vanilla* PPO and their transferred counterparts are shown in Figure 10.

Regarding first the classic UNN and the *vanilla* PPO, we observe that similarly to the Tennis environment, those curves indicate that even though

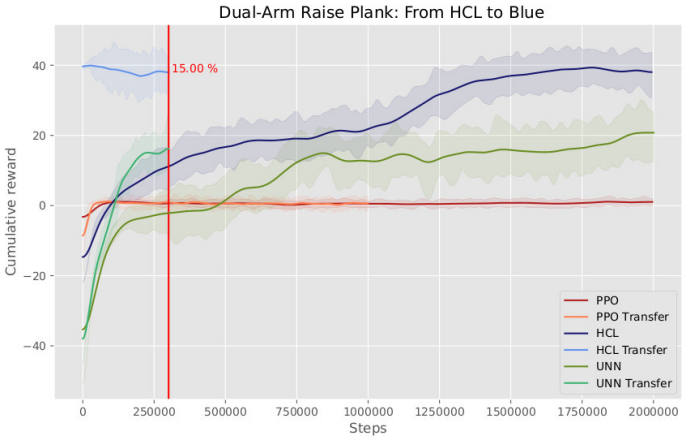


Fig. 10: Learning performances in Dual-Arm Raise Plank environment. PPO and UNN are performed using two Kuka arms, transfer to two Berkeley Blue robots.

the UNN policy starts with a slightly inferior mean reward to the *vanilla* PPO policy, the UNN policy improvement is faster and achieves a much higher mean reward. In particular, neither *vanilla* PPO policy nor its transferred version to a dual Berkeley Blue robot setup are able to succeed in this task. In contrast, the transferred version of the UNN, again controlling two Berkeley Blue robots, even though starting with a low mean reward level, quickly recovers the final performance level. While yielding a non-negligible improvement compared to the *vanilla* approach, the UNN method in this environment cannot match the efficiency of the HCL framework. Indeed, as can be seen in Figure 10, the HCL agent mean reward evolution presents a steeper evolution and is able to achieve two times the classic UNN mean reward, thus also surpassing the *vanilla* approach. Moreover, as for transfer efficiency, it can be seen that the transferred agent immediately starts with a reward level comparable to the one achieved by the HCL policy after 2 million timesteps, effectively showing the transfer efficiency. In this case, this indicates that the transferred policy didn't even need retraining.

At test time, we define the best agent as the one capable of reaching to the plank and keeping it in the proximity of the goal position for the longest time. Again, we average this metric expressed in timesteps over 100 episodes with a time limit of 2000 timesteps. Table 3 confirms the results shown in Figure 10. As a matter of fact, the results obtained by the two UNN configurations, both classic and HCL, are clearly superior to the ones the *vanilla* policy is able to achieve. In this specific case, the HCL policy doesn't even need retraining and the transferred policy achieves a better test score than every other configuration, with the exception of the original HCL policy. The UNN policy also

outperforms the *vanilla* PPO policy, even though transfer is not as efficient as in the HCL case.

Table 3: Dual Arm Raise Plank Summary

		Config.	**Training**	**Performances** (in timesteps)
HCL	Original	HCL	100%	1557.05
	Transfer to	B.Blue-B.Blue	0%	1545.65
	Transfer to	B.Blue-B.Blue	15%	1547.02
	Transfer to	Kuka-B.Blue	0%	1451.03
UNN	Original	Kuka-Kuka	100%	987.11
	Transfer to	B.Blue-B.Blue	0%	4.41
	Transfer to	B.Blue-B.Blue	15%	926.66
PPO	PPO	Kuka-Kuka	100%	139.40
	Transfer to	B.Blue-B.Blue	0%	8.45
	Transfer to	B.Blue-B.Blue	100%	152.54
	Transfer to	Kuka-B.Blue	100%	147.78

The Dual-Arm Raise Plank environment presents various features that are prone to lead a *vanilla* agent to failure and thus provide interesting insights to understand better the HCL framework advantages. In particular, the succession of objectives is rarely trivial to master in reinforcement learning. This is further exacerbated by the number of degrees of freedom in the controlled robot. Using the UNN method abstracts the tasks and lets the agent focus solely on where to place and move its effectors. As for the biped case, it is possible to pretrain the modules in order to include a certain degree of bias and help the UNN policy with these initial conditions. Nevertheless, there are two main points that distinguish this task from the biped task. In particular:

- In the Biped Walking task,
  - the qualitative appeal of the walk cycle depends on the leg configuration and not only on foot placement
  - and from a modelling point of view, it is unclear how to design a HCL version of the task.
- While the Cooperative Raise Plank task as designed
  - isn't affected by the arms configuration, as long as there are no self-collision
  - but does require an enhanced accuracy regarding the effectors control. Thus, unless special care is dedicated to the module training, it is likely that it might hurt the UNN policy learning performances.
  - Finally, the modelling process and constraints setup for HCL training is particularly straightforward.

These points indicate that creating a HCL version of the environment is a valuable alternative. As seen in Figure 10 and Table 3, the HCL version of the UNN behaves in a more desirable way than its counterparts, thus offering

a robot-agnostic model of the task to various robots. Then, if the modules of a particular robot are sufficiently precise, it is possible to succeed in the task with no retraining at all and still yield higher performances than other policies.

Obviously, it is theoretically possible to design a reward function that would lead a *vanilla* PPO policy to walk in a predictable manner or reach and manipulate a plank with a high accuracy. However, this is a very complex task that requires a considerable amount of feedback, tweaking and state representation engineering in order to find the ponderation that would, in the biped case, allow the policy to walk fast while not degrading the other important components too much, namely the steering and the height control. In the manipulation task, the sequential nature of the environment would also call for careful and thorough fine-tuning of the reward and the simulation to ensure that the policy finds a reliable solution. Using the UNN method enables to greatly simplify this design process by embedding in the robot limbs a bias, that acts similarly as soft constraints. Alternatively, setting up a HCL alternative when the conditions concur to it is highly useful to help the policy find a solution in highly complex and non-linear environments.

4.4 Recovering robot modules

So far, the presented applications have focused on cases where the UNN training relies on fixed pre-trained modules to learn the task. Let us now consider the configuration described by the red path in Figure 2. That is, the UNN for a task already exists, but using a new robot with no robot module available. We refer to this already existing UNN as the **original**. As explained in Section 2.2, it is possible to take advantage of the **original** UNN to recreate the robot module. Specifically, two different ways of training the modules may be considered:

- Classic recovery: Leverage the previous MDP for training the modules. That is, the same reward function used to train the task is used to train the agent. In this case, the UNN weights are fixed while the module weights are updated by backpropagation.
- Residual recovery: In this configuration, the agent is penalized for straying away from the UNN instruction, hence considering Equation(15).

$$R_{recovery} = exp(-\alpha|P^T(t_i) - P^R(t_{i+1})|) \tag{15}$$

With α a normalization constant, $P^T(t_i)$ the target effector pose given by the UNN and $P^R(t_{i+1})$ the resulting effector pose.

The Residual approach can be assimilated to a Reaching task for the robot modules, where the target is set at every timestep by the UNN as presented in Figure 11. To demonstrate this method, we consider that the robot modules for the Generic-3 robot are not available, hence must be learned, in the tennis environment, using the UNN learned with the kuka arm, mentioned as the **original** UNN.

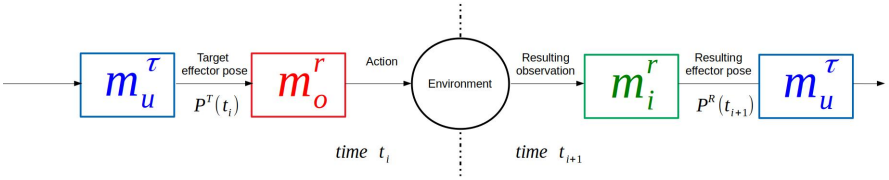


Fig. 11: Representation of the residual recovery process. The module m_u^τ is constant and m_o^r and m_i^r are learned based on the difference between the target and the resulting effector pose.

Figure 12 shows learning progression for the module in the classic recovery case, along with the learning curve of its original (learning of the original UNN with modules pretrained with the reaching task) and the *vanilla* configuration as a baseline. In this case, while the recovered module progressively gets better results, exceeding the *vanilla* approach from the start, its final performance is still below the original UNN even after a training time equal to the original policy.

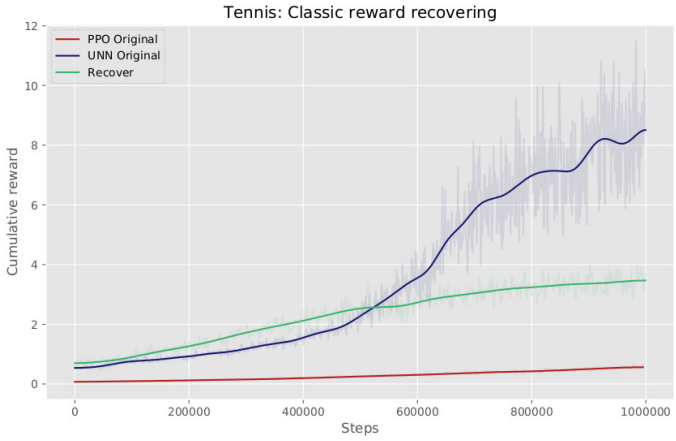


Fig. 12: Module recovery from KUKA to Generic-3: Classic recovery setup for the Tennis environment. The PPO Original and UNN original curve are used as baseline to measure the recovering efficiency

Figure 13 shows results for the residual recovery approach, where the optimal policy should have a mean reward per episode close to 0, because, as previously explained, the residual recovery approach is similar to a reaching task, except that the target is set by the UNN. In this case, the policy reaches its final performance level in about a fifth of the initial training time. We sum up the test performances results in Table 4, again using the number of bounces

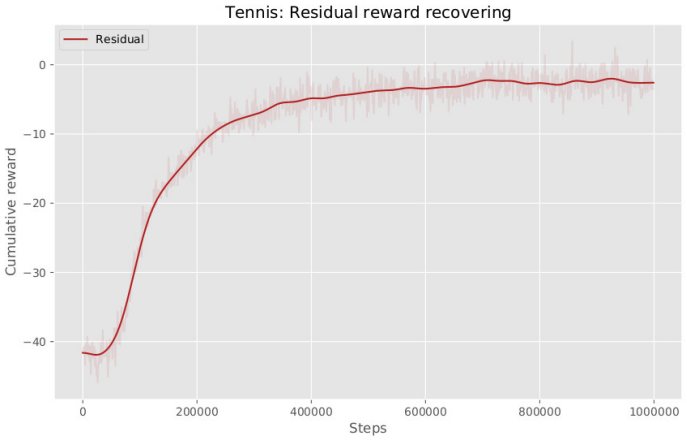


Fig. 13: Module recovery from KUKA to G-3: Residual recovery setup for the Tennis environment.

as the success metric. In this case, the residual recovery approach performance exceeds the classic recovering one while still not doing as well as the original policy. Indeed, the classic recovery approach doesn't provide an improvement signal as clear as the residual recovery one. That is, the output module has to infer a control policy based on its internal perceptions, the UNN output and the reward. However, this results in a tail-chasing configuration, as the UNN relies on the output module itself, making it even more difficult than in the initial setup to collect reward, in particular in the case of sparse rewards. On the other hand, the residual recovery approach doesn't make use of the environment reward to progress, as it uses the environment uniquely as a support for the UNN to guide the output module. To sum up, this experiment shows that it is possible to recover the modules using a pretrained UNN, yielding acceptable, but inferior results than when the workflow is reversed, that is first pre-training the module, and then the UNN.

Table 4: Recovery summary

Robot	**Recovery Type**	**Performances** (in bounces)
Kuka	Original	18.74
G3	Classic	10.87
	Residual	14.50

5 Related Works

Transfer learning is a very common approach in various machine learning fields, such as computer vision and natural language processing. Indeed, due to the amount of computation needed for learning language models [18],[10] or relevant convolution filters [19], there are important advantages in relying on these pretrained models. Furthermore, the architecture of these models does implicitly structure knowledge, in particular in the dimensional bottleneck in the last fully connected layers of a vision/language model. However, in reinforcement learning, common architectures are usually less greedy in terms of computation and do not present any particular bottleneck that infers the role of each layer intuitively. Hence, transfer learning in RL is usually approached with the idea of having an agent being able to do multiple tasks [20], being able to quickly readapt or develop generalized strategies to succeed in different environments [21] or to adapt to the user in case of telemanipulation [22]. Interestingly, we can also note that the usual RL training frameworks such as OpenAI Gym [23] for testing RL algorithms are not dedicated to transferring knowledge. Thus, even if robust algorithms ([16], [15]) exist and are able to succeed in complex, high dimensional non-linear environments, the resulting agent is almost never used beyond its training environment. Among the reasons for this policy concealment, we can point out the fact that RL policies do not generalize well outside of the training distribution, resulting in poor performance when deployed to other environments. Various works aim at mitigating this issue, such as [24], [5] [25] and [26] that leverage domain randomization to help the policy become insensitive to world lighting and texturing, which helps transfer a learned policy for manipulating a cube from a simulated environment to the real-world. Another approach is Residual Learning, successfully used in [27] where Convolutional networks are used along with the ballistic equation for throwing precisely objects of diverse shapes and weight distributions in specified bins. Despite the impressive results of these works, the learned control policies are completely tied to the robot parameters and cannot be reused by another robot of a different configuration, which is exactly what the UNN approach offers to do.

The UNN method can also be assimilated to a hierarchical/compositional approach of reinforcement learning, given that we first master basic skills and then use them to complete more complex tasks. This path has been taken by a number of recent works such as [28] where a hierarchical architecture is setup. In this case, a master policy selects a controller for a certain number of timesteps from a set of policies, similar to primitives. While providing a useful set of learned controllers after training, the networks all share the same parameters which makes reusability uneasy. Compared to the HCL, the use of simple model during learning phase was already explored in [29] for brachiation motion to facilitate the learning of the policy for the full model. Considering simulated robots and actual robots, the UNN may also refer to sim-to-real transfer techniques and was already extended to perform simple planar application while dealing with delay issue [30].

6 Conclusion and Prospects

In this paper, we presented a broad range of applications for the proposed Universal Notice Network (UNN) approach, a method that disassociates the agent control logic from the task it is supposed to accomplish. For each detailed environment, we compared our technique with a classical state-of-the-art algorithm and we demonstrate the benefits and advantages of our method, both in terms of learning speed and final performance. We further tackle the issue of transfer learning in RL in a way that is seldom regarded in recent works. Considering seven different tasks and five different robot kinematic morphologies, we show that our method enables a more efficient transfer than current methods, allowing an agent to perform as well as the original policy without retraining at all, in some cases. The experiments set up allowed us to detect that the classic UNN architecture may convey a bias from the modules pre-training which can in various situations be used as an advantage. For configurations in which the bias can be detrimental or simply undesired, we also proposed an experimental framework, the Heuristic Curriculum Learning, for overcoming such biases.

Future works will investigate more deeply the modules bias and determine how best to use it. Also, we plan to focus on a way to embed soft constraints within the modules in order to study how the learning and exploration process of a policy can be altered with this pretraining step. Eventually, here we set analytical kinematic models for the state representation, we plan to explore the use of neural network as robot basis to look for a suitable state representation.

References

[1] Cunha, J., Serra, R., Lau, N., Lopes, L.S., Neves, A.J.R.: Batch Reinforcement Learning for Robotic Soccer using the Q-batch Update-Rule. Journal of Intelligent & Robotic Systems **80**(3), 385–399 (2015)

[2] Mnih, V., et al.: Playing Atari with Deep Reinforcement Learning. CoRR **abs/1312.5602** (2013)

[3] Peng, X.B., Kanazawa, A., Malik, J., Abbeel, P., Levine, S.: SFV: Reinforcement learning of physical skills from videos. ACM Trans. Graph. **37**(6) (2018)

[4] Li, Y., Ni, P., Chang, V.: Application of deep reinforcement learning in stock trading strategies and stock forecasting. Journal of Intelligent & Robotic Systems, 283–300 (2019)

[5] Andrychowicz, O.M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., Zaremba, W.: Learning dexterous in-hand manipulation. The International Journal of Robotics Research **39**(1), 3–20 (2020)

[6] Silver, D., *et al.*: Mastering the game of go without human knowledge. Nature **550**, 354–359 (2017)

[7] Sun, Y., Cheng, J., Zhang, G., Xu, H.: Mapless motion planning system for an autonomous underwater vehicle using policy gradient-based deep reinforcement learning. Journal of Intelligent & Robotic Systems **96** (2019)

[8] Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31 (2018)

[9] Srinivas, A., et al.: Universal planning networks. CoRR **abs/1804.00645** (2018)

[10] Howard, J., Ruder, S.: Fine-tuned language models for text classification. CoRR **abs/1801.06146** (2018)

[11] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track **9**, 249–256 (2010)

[12] Mounsif, M., et al.: Universal Notice Network: Transferable Knowledge Among Agents. 6th 2019 International Conference on Control, Decision and Information Technologies (IEEE-CoDIT 2019) (2019) [cs.RO]

[13] Juliani, A., et al.: Unity: A general platform for intelligent agents. CoRR **abs/1809.02627** (2018)

[14] Starke, S., et al.: Evolutionary multi-objective inverse kinematics on highly articulated and humanoid robot. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (2017)

[15] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017)

[16] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1861–1870 (2018)

[17] Mounsif, M.: Exploration of teacher-centered and task-centered paradigms for efficient transfer of skills between morphologically distinct robot. PhD thesis, Université Clermont Auvergne, Ecole doctorale des sciences pour l'ingénieur de Clermont Ferrand. (2020)

[18] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (2019)

[19] He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(2), 386–397 (2020)

[20] Eysenbach, B., Gupta, A., Ibarz, J., Levine, S.: Diversity is all you need: Learning skills without a reward function. In: International Conference on Learning Representations (2019)

[21] Jaderberg, M., Czarnecki, W.M., Dunning, I., Marris, L., Lever, G., Castañeda, A.G., Beattie, C., Rabinowitz, N.C., Morcos, A.S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J.Z., Silver, D., Hassabis, D., Kavukcuoglu, K., Graepel, T.: Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. CoRR **abs/1807.01281** (2018)

[22] Tao, L., Bowman, M., Zhou, X., Zhang, J., Zhang, X.: Learn and transfer knowledge of preferred assistance strategies in semi-autonomous telemanipulation. Journal of Intelligent & Robotic Systems **104**(48) (2022)

[23] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. CoRR **abs/1606.01540** (2016)

[24] Cao, Z., Kwon, M., Sadigh, D.: Transfer reinforcement learning across homotopy classes. IEEE Robotics and Automation Letters **6**(2), 2706–2713 (2021)

[25] Zhao, W., Queralta, J.P., Westerlund, T.: Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 737–744 (2020)

[26] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: 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), pp. 23–30 (2017)

[27] Zeng, A., Song, S., Lee, J., Rodriguez, A., Funkhouser, T.: Tossingbot: Learning to throw arbitrary objects with residual physics. IEEE Transactions on Robotics **36**(4), 1307–1319 (2020)

[28] Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, J.: Meta-learning shared hierarchies. In: International Conference on Learning Representations (2018)

[29] Reda, D., Ling, H.Y., van de Panne, M.: Learning to brachiate via simplified model imitation. In: ACM SIGGRAPH 2022 Conference Proceedings. SIGGRAPH '22 (2022)

[30] Beaussant, S., Lengagne, S., Thuilot, B., Stasse, O.: Delay aware universal notice network: Real world multi-robot transfer learning. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1251–1258 (2021)

Declarations

Acknowledgments-funding. This work has been sponsored by the French government research program Investissements d'avenir through the RobotEx Equipment of Excellence (ANR-10-EQPX-44) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the program of Regional competitiveness and employment 2007-2013 (ERDF – Auvergne region), by the Auvergne region and by French Institute for Advanced Mechanics.

Conflict of interest/Competing interests. The authors have no relevant financial or non-financial interests to disclose."

Authors' Contribution. The technical implementation and the first draft manuscript was written by Mehdi Mounsif and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript

Additional declarations.

- Ethics approval : Not applicable
- Consent to participate : Not applicable
- Consent for publication : Not applicable
- Availability of data and materials : Not applicable
- Code availability : Not applicable

Appendix A Additional results

In this section, additional results for the presented environments are detailed. Specifically, we show both learning curves and test performances for the Raise Plank, Cooperative Raise Plank and Basket-Ball task. These curves confirm the observations detailed in Section 4, especially concerning the transfer efficiency. Indeed, while in the Raise Plank and Cooperative Raise Plank, the *vanilla* PPO policy and the UNN exhibit a similar final mean reward, see Figure A1 and A2, the transferred PPO policy is in these cases negatively

impacted and its learning progress is undeniably slow, if not null. However, the transferred UNN policy starts with a superior mean reward and, for the Raise Plank and Cooperative Raise Plank environment, reaches the final mean reward of the previous policy in a fraction of the total time. At test time, we use the following metrics, averaged over 100 episodes:

- **Raise Plank**: A counter is incremented for each timestep if the plank altitude is over a height threshold
- **Cooperative Raise Plank**: A counter is incremented for each timestep if the plank is in the proximity of a target position
- **Basket-Ball**: A counter is incremented for each timestep if the ball is kept over a height threshold

These results detailed in Table A1 are coherent with the final mean reward of the learning curves and further emphasize the fact that transfer learning using the UNN approach outperforms by an important margin straightfoward transfer. As Table A1 refers in some cases to the Generic 2 (G2) robot, we inform the reader that this robot architecture is similar to the G3 robot, with the slight difference that a segment was removed.

Table A1: Test Performances Summary

		Config	Training	Performances (in timesteps)
Raise Plank				
	Original	Kuka	100%	805.70
	Transfer to	Generic 2	0%	244.64
UNN	Transfer to	Generic 2	30%	723.65
	Original	Kuka	100%	755.02
	Transfer to	Generic 2	0%	20.57
PPO	Transfer to	Generic 2	100%	80.81
Cooperative Raise Plank				
	Original	Generic 3-Generic 3	100%	552.68
	Transfer to	G3-G2	0%	111.20
UNN	Transfer to	G3-G2	30%	547.88
	Original	G3-G3	100%	589.96
	Transfer to	G3-G2	0%	3.57
PPO	Transfer to	G3-G2	100%	8.08
Basket-Ball				
	Original	HCL	100%	1745.04
	Transfer to	Kuka-Kuka	0%	749.58
UNN	Transfer to	Kuka-Kuka	30%	1022.17
	Original	G3-G3	100%	807.54
	Transfer to	Kuka-Kuka	0%	1.52
PPO	Transfer to	Kuka-Kuka	100%	478.85

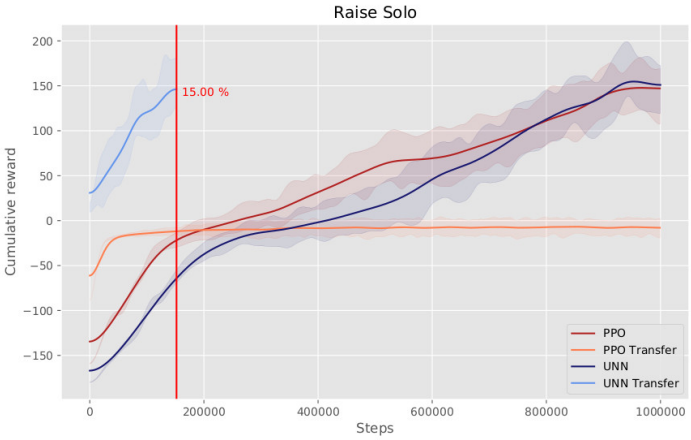


Fig. A1: Addition learning curves for Raise Plank learning performances and transfer from KUKA to G2

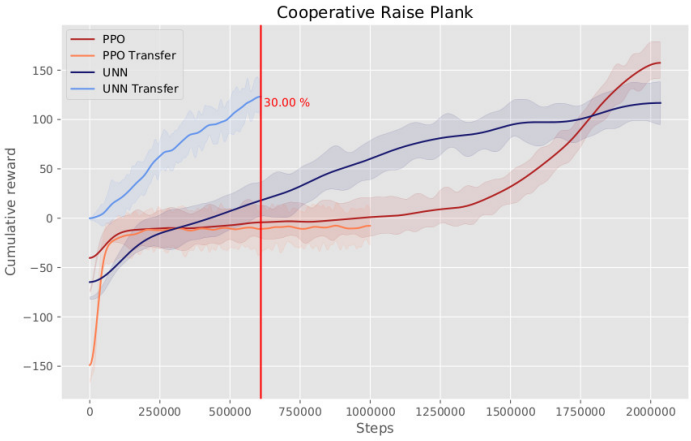


Fig. A2: Addition learning curves for Cooperative Raise Plank learning performances and transfer from G3-G3 to G2-G3

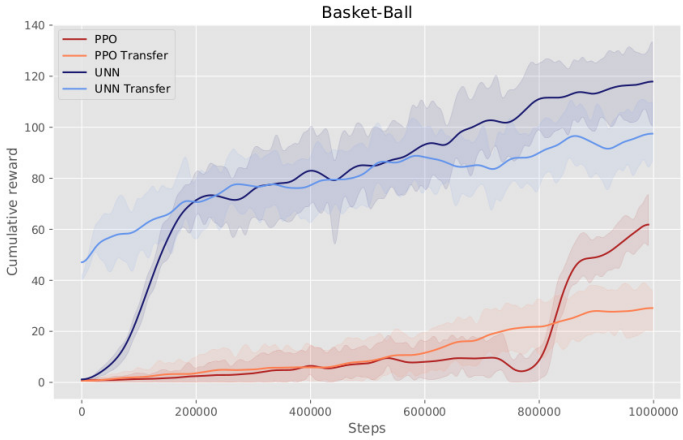


Fig. A3: Addition learning curves forBasket-Ball learning performances and transfer from HCL to KUKA-KUKA