Universal Notice Network: Transferable Knowledge Among Agents

Mehdi Mounsif, Sebastien Lengagne, Benoit Thuilot, Lounis Adouane

Universal Notice Network: Transferable Knowledge Among Agents

Mehdi Mounsif, Sebastien Lengagne, Benoit Thuilot and Lounis Adouane

***Abstract*— Being able to learn and transfer skills from one agent to another is a fundamental feature in constructing even more intelligent behaviors. In this paper, we introduce a new kind of architecture and information pipeline that aims to enable the transmission of skills from one robot to one or several others. The Universal Notice Network (UNN) originality lies in the fact that it clearly distinguishes knowledge necessary to solve the task from the agent intrinsic perceptions and capabilities, hence increasing its reusability and its potential transmission to other agents. In various experiments, focusing on manipulation and comanipulation tasks in original environments, we demonstrate the capabilities of the proposed method that takes advantage of reinforcement learning algorithms and domain knowledge, such as forward geometric model and inverse kinematics. In particular, we show that a learned UNN through the interactions of an agent with its environment is transmissible to other agents, conserving a similar perfomance level.**

I. INTRODUCTION

From any point of view, fine-grained and versatile control of robot would yield great benefits. Various research fields focus on this goal: control, planning, and more recently learning-based approaches have been particularly visible among the research community. Combining modern deep learning techniques and reinforcement learning principles has led to impressive results. Indeed, both model-based and model-free methods demonstrate high performance in complex tasks such as the game of go [18], [19] or continuous control [9] [13] and [14]. Even though recent works show reflexion and progress toward adaptable skills or knowledge, in particular using character retargeting [15] or environment adaptation [22], most agents are usually trained on a specific environments, with no option for transferring skills or gained knowledge. Whenever the environment requires the agent to learn reusable skills, it would be highly appreciable to be able to embed the logic necessary to solve the task in a separate module so it is potentially transferable to other entities.

This concept is more fully understood if we focus on the industrial use of robots. In this case, there are two main incentives to rely on such a methodology. The first one corresponds to the case where a robot could easily be replaced by another, with more or less different specifications (different number of joints, segment length,...), as depicted in Figure 1. Another motivation behind this approach is life-long adaptation. In fact, while the instructions given to the robot might be the same during its whole lifetime (e.g., pick and place, reach, push, ...) the robots component will be altered by its use, changing its model over-time. The proposed methodology could adapt to these modifications and be able to deal with damages either due to shocks or age decay.

All authors are with Université Clermont Auvergne, CNRS, SIGMA Clermont, Institut Pascal, F-63000 Clermont-Ferrand, France. mehdi.mounsif@uca.fr

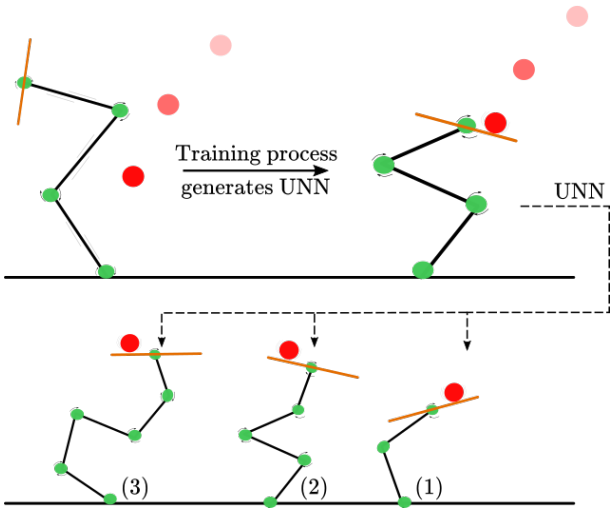


Fig. 1: Transferring learned Universal Notice Network (UNN) to other manipulator structures

Next section introduces relevant works to our current research and emphasizes areas where improvements in transfer learning can be made. Section III presents the main concepts and contributions of this paper, leading to section IV where we demonstrate the capabilities of our method in various experiments. Finally, section V focuses on leads of improvements and discussions.

II. RELATED WORKS

In the robotics field some works describe how to control robots based on a hierarchical decomposition of tasks into a set of simpler sub-tasks with some priorities [10]. The high priority sub-tasks ensure the integrity of the robot and of its environment, the low priority tasks describe the action to achieve. The desired task will be performed if it does not conflict with the high priority sub-tasks. Unfortunately, this method requires a model of the robot and a mathematical description of the action to be performed, and of its derivative regarding the robot joint positions. Except for specific cases, it is usually not trivial to frame a robotic problem in such a way. Learning based techniques, however, are generally not impacted by these constraints. For instance, the Dactyl system by OpenAI [11], operating on the Shadow Hand was trained on simulation and yet is able to perform complex manipulation tasks of rigid objects in the real-world. In [15], a model-free method learns to control an articulated ragdoll

in a simulator to mimic acrobatic human motion with an end-to-end pipeline involving recovering 3D poses images with no depth information and reconstructing the human body joints trajectories.

Commonly, reinforcement learning agents are trained with the only objective of performing the task they are taught through the reward function. In a few cases, this agent can be used as an expert to provide demonstrations, for instance in [6], which can be seen as an instance of Imitation Learning. But in many works, the gained knowledge doesn't go any further than its initially destined task. Transferring knowledge has mostly been an issue the reseachers working with deep convolutional networks [4] or NLP models [7] had to deal with. Indeed, given the resources needed to train a model, many works detail methods aiming at generalizating a model in a setting different of the one it has learned with. This trend is also perceptible in reinforcement learning, with techniques such as domain randomization [21]. Also, to match the IMPALA architecture [2], DeepMind proposed [1] a corpus of environments for agents navigation, featuring mazes with walls of differents colors and textures for benchmarking transfer of skills from one environment to another. While partially addressing the transfer of skills between environments using a single agent, the knowledge gained through learning in these models isn't distangled, e.g., it is not possible to interpret and determine regions of the model where the task is solved. The idea of transferring knowledge in a concise and distangled-enough way to be usable for various tasks isn't new in the learning community.

Hence, the most salient drawback with current transfer learning settings may be the fact that no precaution is taken to enforce knowledge in a specific part of the model. This, in turn, implies that it is not possible to isolate parts of the model and transferring them to other agents.

Another promising approach, yet less common, is Meta-Learning. In this setting, a model is trained on a corpus of tasks with the objective of finding a model initialization point in the parameters space where it is only a few iterations away from a local optimum, ie: where it will produce a good generalization concerning a specific task. Recent notable work include the MAML algorithm [3], where the authors introduced the interleaved training procedure, composed of an inner-loop for task-specific optimization and outer-loop for a more general gradient. Further developments such as the CAML algorithm [24] decrease the number of adjustable parameters and improve the overall performance of the model. While the formulation is appealing, the results presented focused on trivial tasks, such as regression over sine curves, or in the reinforcement learning framework, moving a particle towards a point in a bidimensional space, which is rather far from robotic task in an industrial context.

Overall, most of the transfer techniques are rather aiming at transferring skills gained through interactions with a specific environment into another one, increasing the agent adaptability and retargeting their abilities to new domains. However, to the best of our knowledge, there is no specific way to ensure that knowledge necessary for solving a task is not spread all over the layers, making it then difficult to transfer blocks between agents. The following section introduces our novel method, the Universal Notice Network that adresses exactly this important drawback.

III. Universal Notice Network

A. Problem formulation

We start by introducing several notations, useful for the problem formulation: $\mathcal{T}$, the task to be solved, R is one robot physically able to perform the task and s is a vector of observations coming from the environment. Thus, we are seeking a controller $\mathcal{C}$ such that following sequences of actions given by:

$$a = \mathcal{C}(\mathcal{T}, s, R) \tag{1}$$

can ultimately solve the task.

B. Concept

The proposed Universal Notice Network (UNN) objective is to provide a dedicated task module that can be set in the middle of a control pipeline, allowing any compatible robot to solve the task. To motivate our approach we start with an example, depicted in Figure 2. Let us consider two persons charged each one of moving a heavy load from a point to another. The initial and current target location are given on a notice, that they share. In this setting, it is straightforward to see that both individuals have the same task description but with different physical capabilities. In other words, while their environment is the same, they might end up choosing various ways of completing the task.

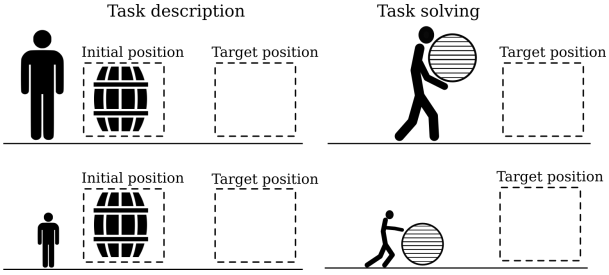


Fig. 2: Solving the same task with different capabilities. Each row shows a possible outcome based on individual strengths and preferences.

C. Formalism

Our aim is to show that it is possible to construct a central bloc that holds sufficient knowledge to solve a task. This bloc, the UNN, must be independant from the agents physical capabilities so that it is transferrable to other entites. However, different bodies configurations imply different intrinsic observation e.g., the vector holding this information will have as many dimension as needed to describe the entity. To deal with this particularity we rely on the following mechanism: we do not feed the full observation vector s from the environment into the UNN. Rather, the observation is split into a task observation s_t and an intrinsic observation vector

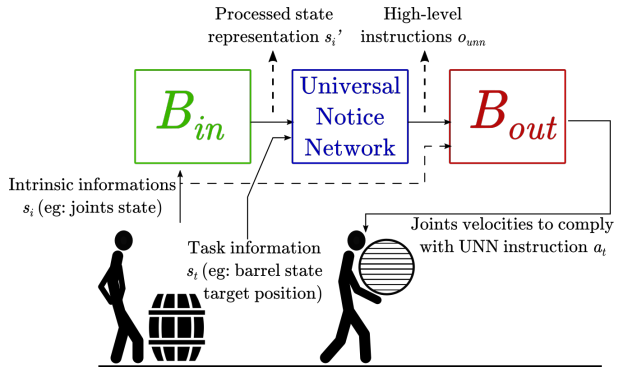


Fig. 3: UNN architecture. The agent receives an observation vector s, splitted into a robot observation vector s_i processed by the input base and a task observation vector s_t. The UNN, operates on the processed robot information and the task informations. Later the output base, computes the action based on UNN output c and robot intrinsic state vector s_i

that holds data concerning the robot s_i, $s = \{s_i, s_t\}$. The latter is processed by an input base $\mathcal{B}^R_{in}$, a model specific to the robot R that translate s_i into a vector s_i', understandable by the UNN. The UNN hence receives as input s' such as:

$$s_i' = \mathcal{B}^R_{in}(s_i) \tag{2}$$

$$s' = \text{concat}[s_i', s_t] \tag{3}$$

Based on this vector s', the UNN model outputs a high-level instruction, called o_{UNN}. However, in the same manner as before and in order to keep the UNN independant of structures considerations, we introduce an additional model, the output base $\mathcal{B}^R_{out}$, again, specific to each agent. This output base uses the intrinsic robot observation s_i to translate o_{UNN} into an action suitable with the robot configuration.

$$o_{\text{UNN}} = \text{UNN}(s') \tag{4}$$

$$a = \mathcal{B}^R_{out}(o_{\text{UNN}}, s_i) \tag{5}$$

Finally, the whole process, as shown in Figure 3 can be synthetized as:

$$a = \mathcal{B}^R_{out}(\text{UNN}(\mathcal{B}^R_{in}(s_i), s_t), s_i) \tag{6}$$

D. Background in Reinforcement Learning

While in principle this technique is compatible with any kind of learning approach, we rely on reinforcement learning to train the UNN. The usual reinforcement learning framework involves various concepts: we consider an agent interacting with an environment. The environment is defined as a Markov Decision Process $\mathcal{M} = [\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma]$ where $\mathcal{S}$, $\mathcal{A}$ are respectively the set of states and actions available within the environment. $\mathcal{P}$ is the probability transition between s_{t+1} and s_t when selecting action $a \in \mathcal{A}$ ie: $p(s_{t+1}|s_t, a_t)$. $\mathcal{R}$ is the reward function, reflecting the desirability of the state reached and γ is the discount factor for future rewards, which goal is to favorise immediate rewards.

In reinforcement learning, the objective is to find the policy that maximizes the expected cumulative reward: $\pi^\star_\theta = \underset{\pi_\theta}{\text{argmax}}\ \underset{\tau\sim\mathcal{D}}{\mathbb{E}}[\sum_{t=0}^{T}\gamma^t r_t]$, where $\mathcal{D}$ is a set of fixed-horizon trajectories, defined as a set of states-actions pairs $\{(s_0, a_0), (s_1, a_1), ..., (s_n, a_n)\}$. When using neural networks as function approximators, we actually seek the best policy parameters θ.

The RL community yields various methods suitable for finding the optimal policy [5], [23]. When dealing with continuous spaces, it is common to take advantage of policy gradient methods, rather than Q-Learning that would not be as efficient due to the curse of dimensionality [20]. However, simple policy gradient agents are very brittle and fail to improve their performance due to gradient variance issues. Instead, this work is based on a Policy Gradient variant called Proximal Policy Optimization [16], capitalizing on an actor-critic and trust-region setting. These two concepts improve strongly the agent performance thanks to reduced variance in the performance estimation and a controlled policy optimization step. Policy gradients algorithms use gradient ascent to maximize the policy performance, estimated using rollouts of the policy in the environment. The most communly used estimator has the following form:

$$\hat{g} = \hat{\mathbb{E}}[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t]$$

where $\hat{A}_t$ measures how good the action was and, in the simplest policy gradient theorem [20] is the plain reward r_t. Actor-Critic methods use a supplementary network, called the critic to estimate the state value $V(s_t)$, corresponding to the expected cumulative reward from that state until the end of the episode, which in turn enables estimating the advantage such as $\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$. Finally, trust-region methods aim at taking the biggest optimization step without destroying the policy. In this view, the PPO algorithm maximizes a clipped surrogate objective:

$$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)$$

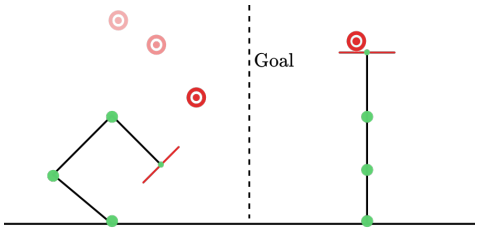
where $r(\theta)$ is the probability ratio: $r(\theta) = (\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)})$ between the policy parametrized by its current weights π_θ and a previous version $\pi_{\theta_{old}}$.

IV. Experiments

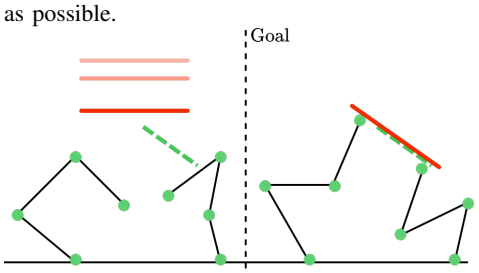
We chose to implement the learning environments through the Pymunk physic library, a python API relying itself on the open-source Chipmunk engine [17]. For the learning part, the PyTorch [12] learning library was used and the baseline PPO implementation is [8]. Videos of trained agents are available at `https://bit.ly/2QdOIep`.

A. Experiments configuration

As stated previously, we expect the UNN to act as a plug-and-play module by being seamlessly transferred between various agents. In this purpose, we design the various experiments cases to demonstrate our method. Let



(a) The *catcher* task: A ball is dropped or thrown and the agent has to catch it and raise it as high as possible.



(b) The *double-catcher* task: A bar is dropped while the a target position and orientation is specified to the agent(the dashed line)

Fig. 4: Two tests environments

the notation R_j^i describe a robot with j degrees of freedom and belonging to the sub-case $i \in \{A, B, C, D\}$. The sub-cases explicits the base types. A corresponds to the case where the bases are analytical. Specifically, we use forward kinematic model to compute the robot effector position and orientation of the effector base. The output base relies on inverse kinematics to compute joints velocities to comply with instructions given by the UNN. B is used for cases where bases are learned models of the analytical approaches, obtained using supervised learning separately. C and D cases however indicate that the UNN has already been learned and the bases are retrieved through training. The difference between these two subcases is that C is initialized before starting the training while D indicates that the bases are initialized with completely random weights. The baseline PPO implementation without bases, e.g., mapping states directly to joints speed, will be noted R_j.

B. Manipulation tasks

The first environment, called *catcher* consists in a multi-joint robotic arm with a bar attached to its effector and a ball, falling from a higher position with an initial velocity, see Figure 4a. Each episode starts with setting the robot in a random configuration. The episode ends if the ball height is below a threshold, implying that it has either fallen from the robot's bar or that the agent was not able to hold it, or after 500 steps. The goal of this environment is for the robot to catch the ball and raise it as high as possible. To incentivize the agent to fulfill the task, we introduce the following reward function: $R = c \times h$, where c is a binary operator that is 1 only if there is a contact between the agent effector and the ball, otherwise 0, and h is the ball height.

In this setting, we first assess the learning performances

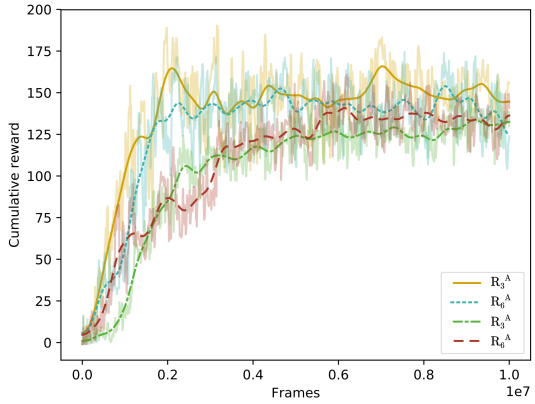


Fig. 5: Cumulative reward over training for the *catcher* task.

of our method using a baseline PPO algorithm. Figure 5 shows the evolution of the cumulative reward along training for various robots R_3^A, R_6^A, R_3, R_6. Cumulative reward is a frequently used metric to benchmark reinforcement learning algorithms. It depends on the environment reward magnitude and represents the performance of an agent through one episode. The higher the cumulative reward, the better the agent performed. As can be seen in Figure 5, the performances of the UNN agents reaches higher cumulative rewards while also being slighlty faster than the baseline PPO agents. Once the UNN is trained, the final performances are evaluated through 2000 episodes on the tasks. Each of these episodes starts with setting the initial condition to some preset determined beforehand in order to have uniform test conditions. Along these runs, if the policy holds the ball for more than 450 steps out of 500 we consider that it has succeeded, otherwise the episode is counted as a failure. Table I shows comparaisons between a baseline PPO agent and a UNN agent for various robot configurations. Given that the ball initial velocity or robot initial configuration may not allow the agent to succeed in the task, the table displays common successes (e.g., both the UNN and PPO reached at least 450 steps), common failures and cases where only one of them succeeded. We can see that in this case the UNN performances are competitive with the PPO baseline. While these results fit with what has been observed during multiple training session and that PPO is among the most stable and consistent RL algorithms, it is important to keep in mind that reinforcement learning performances can vary between two training sessions and that these results could be different for another seed.

C. Transmission of a trained UNN

We now place ourselves in the case where we have in our possession a trained UNN, from last experiment. We demonstrate here that it is possible to conserve a similar level of performances when transfering a trained UNN to another manipulator structure. If the bases are of type A or B, this can be done directly. Using C of D bases types is

PPO R_i vs UNN R_i^A

	Common result		Distinct result	
Config	Success	Fail	Success PPO	Success UNN
3 joints	521	441	362	676
4 joints	594	397	521	488
5 joints	516	472	454	558
6 joints	416	572	511	501

TABLE I: Comparative results on *catcher* of a baseline agent and UNN with configuration A: analytical bases. Statistics are obtained after 2000 episodes

PPO R_i vs UNN R_i^B

	Common result		Distinct result	
Config	Success	Fail	Success PPO	Success UNN
3 joints	526	456	685	333
4 joints	605	392	513	490
5 joints	513	511	505	471
6 joints	406	578	509	507

TABLE II: Comparative results on *catcher* of a baseline agent and UNN with configuration B: supervised model of analytical bases

also possible by training only the base. This configuration relates to following case: the forward model is known and we want to recover the inverse model. For these runs, the forward model architecture consists of two layers, with 30 units each, with a tanh linearity in between. The inverse model architecture is a single layer of 64 neurons, followed by a tanh activation function that is finally used to compute Gaussians distributions using a diagonal covariance matrix parametrized by a vector of size n, where n is the number of joints, trained using PPO. To test the transfer aspect, we compare performances of a trained UNN on a specific robot, obtained with type A configurations, given in the first column with a different manipulator structure and base, given in the second column. The same episodes initialization as table I are used. As can be seen in table III, results are consistent between robots. In particular, we observe a higher number of shared successes, which means that the knowledge and methodology is indeed transfered. Imperfect learning, for instance when recovering bases in R_i^C cases, may hinder performances.

Configuration		**Comparaison with R_i^A**			
		Common result		Distinct result	
UNN	Transmitted to	Success	Fail	R_i^A	UNN
R_4^A	R_3^A	1032	548	221	199
	R_3^C	987	414	408	157
	R_5^A	884	521	241	354
	R_5^B	857	528	329	286
	R_5^C	826	507	419	248
R_5^A	R_3^A	1102	497	186	215
	R_3^C	918	454	422	206
	R_4^A	1009	474	197	320
	R_4^C	923	507	308	262
	R_5^C	925	560	332	183

TABLE III: Results on 2000 episodes of UNN transmission to various configurations and base types.

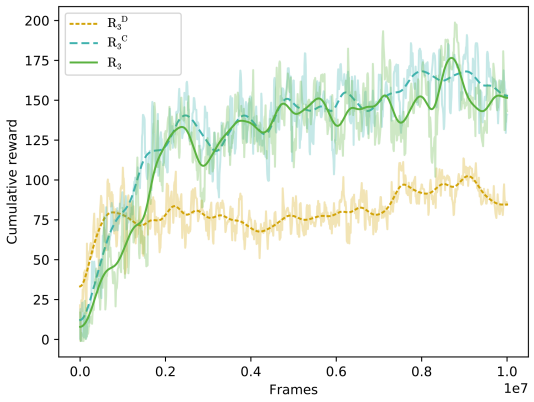


Fig. 6: Comparaison of the evolution of cumulative reward for a baseline agent, and two UNN configurations: R_3^C and R_3^D

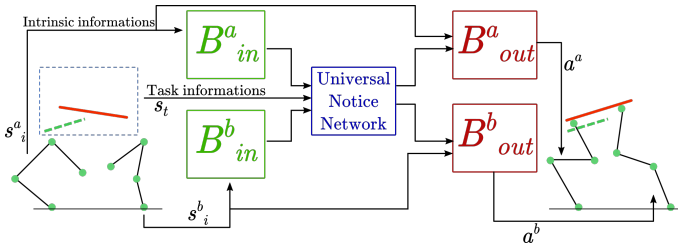


Fig. 7: Principles of the *double-catcher* task where each base can be of a specific type

D. Co-manipulation tasks

To further demonstrate the adaptability of our method, we introduce another environment, *double-catcher*. It consists of two robotic arms operating to catch a falling bar. The objective is to move the bar to a specific position and orientation defined each time a new episode starts. In the same fashion as in the previous environment, the episode ends after a fixed number of timesteps or if the bar falls below a threshold. The reward is also similar, with the following formula being used: $R = c_1 \times c_2 \times d_m \times d$, with C_1, C_2 being the binary contact indicators, d_m a distance mask, that nullifies the reward if the bar position and orientation is too far from the target and $d = \frac{1}{d_{pos}+d_{ori}}$ a value proportional to the inverse distance to target 4b, lower half for a more visual description of this tasks.

Figure 8 shows the evolution of the cumulative reward for two R_3^A controlled either by the UNN or a baseline PPO. On this specific case, the UNN outperforms the baseline by a important margin, both in learning speed and final performance. In table IV, we tested several manipulators structure with different bases types and compared the test results with a baseline. The first two columns hold information about manipulators structure, while the next two columns give the base type. Finally, the two rightmost columns details statistics obtained over 1500 episodes, showing again the adaptability of the UNN approach.

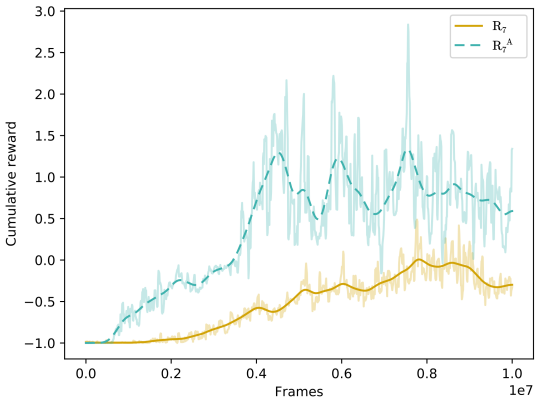


Fig. 8: Cumulative reward over training on *double catcher*

Arm Structure		Base type		Rewards	
Robot 1	Robot 2	Robot 1	Robot 2	Max	Mean
3	3	Baseline		7.45	0.37
3	3	A	A	17.68	3.54
3	4	B	B	14.27	2.39
4	4	A	C	14.89	3.04
4	5	C	C	13.87	2.87

TABLE IV: Comparative results for *double-catcher*

V. Conclusion & Prospects

This paper adresses the problem of knowledge transfer between agents. A novel plug-and-play architecture is presented to prevent learned models to be unusable for other agents. Its key contribution lies in the fact that knowledge is clearly separated from low-level controllers. Thus, this pipeline allows to create separable tasks models that can be easily shared between agents. Simulation results prove our method consistency between robots configurations and competitivity with respect to state-of-the-art baselines. The UNN however preserves generated knowledge for further use and transmission, as shown in multiple benchmark cases. Future work will focus on improving state representation in order to free the UNN from domain knowledge.

Acknowledgment

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.

References

[1] Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., and Petersen, S. DeepMind Lab. *ArXiv e-prints* (Dec. 2016).

[2] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *ArXiv e-prints* (Feb. 2018).

[3] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv:170303400v3* (2017).

[4] Gatys, L. A., Ecker, A. S., and Bethge, M. A Neural Algorithm of Artistic Style. *ArXiv e-prints* (Aug. 2015).

[5] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *ArXiv e-prints* (January 2018).

[6] Ho, J., and Ermon, S. Generative adversarial imitation learning. *CoRR abs/1606.03476* (2016).

[7] Howard, J., and Ruder, S. Universal Language Model Fine-tuning for Text Classification. *ArXiv e-prints* (Jan. 2018).

[8] Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr, 2018.

[9] Lilicrap, T., Hunt, J., Pritzel, A., Hess, N., Erez, T., Silver, D., Tassa, Y., and Wiestra, D. Continuous control with deep reinforcement learning. *arXiv* (February 2016).

[10] Mansard, N., Khatib, O., and Kheddar, A. A unified approach to integrate unilateral constraints in the stack of tasks. *IEEE Transaction on Robotics 25*, 3 (June 2009), 670–685.

[11] OpenAI, :, Andrychowicz, M., Baker, B., Chociej, M., et al. Learning Dexterous In-Hand Manipulation. *ArXiv e-prints* (Aug. 2018).

[12] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W* (2017).

[13] Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ArXiv e-prints* (Apr. 2018).

[14] Peng, X. B., Berseth, G., Yin, K., and van de Panne, M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017) 36*, 4 (2017).

[15] Peng, X. B., Kanazawa, A., Malik, J., Abbeel, P., and Levine, S. SFV: Reinforcement Learning of Physical Skills from Videos. *ArXiv e-prints* (Oct. 2018).

[16] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR* (2017).

[17] Scott, L. Chipmunk2d, 2013.

[18] Silver, D., et al. Mastering the game of go with deep neural networks and tree search. *The Journal of Nature* (October 2015).

[19] Silver, D., et al. Mastering the game of go without human knowledge. *The Journal of Nature* (April 2017).

[20] Sutton, R., and Bardo, A. G. *Reinforcement Learning: An Introduction - 3rd Edition*. MIT Press, 2017.

[21] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *ArXiv e-prints* (Mar. 2017).

[22] Trapit, B., Jakub, P., Szymon, S., Ilya, S., and Igor, M. Emergent complexity via multi-agent competition. *arXiv - OpenAI Technical Report* (2017).

[23] Volodymyr, M., et al. Play atari with deep reinforcement learning. Tech. rep., 2013.

[24] Zintgraf, L. M., Shiarlis, K., Kurin, V., Hofmann, K., and Whiteson, S. CAML: Fast Context Adaptation via Meta-Learning. *ArXiv e-prints* (Oct. 2018).