



HAL
open science

Transfer learning between robots with state abstraction

Samuel Beaussant

► **To cite this version:**

Samuel Beaussant. Transfer learning between robots with state abstraction. Automatic. Université Clermont Auvergne, 2023. English. NNT: . tel-04598434

HAL Id: tel-04598434

<https://hal.science/tel-04598434v1>

Submitted on 3 Jun 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



UNIVERSITÉ CLERMONT AUVERGNE
ECOLE DOCTORAL DES SCIENCES POUR L'INGÉNIEUR DE
CLERMONT-FERRAND

Transfer Learning between robots with State Abstraction

THÈSE

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité

Électronique et Système

présentée et soutenue publiquement par

Samuel BEAUSSANT

le 20 Septembre 2023

JURY :

Serena IVALDI	Chargée de recherche, INRIA Nancy Grand-Est	Examineur
David FILLIAT	Professeur, ENSTA Paris	Rapporteur
Stéphane DONCIEUX	Professeur, ISIR	Rapporteur
Olivier STASSE	Directeur de recherche, CNRS/LAAS	Directeur de thèse
Benoît THUILOT	Maitre de Conférences, Institut Pascal	Co-encadrant de thèse
Sébastien LENGAGNE	Maitre de Conférences, Institut Pascal	Co-encadrant de thèse

List of Figures

1.1	German speaker trying to convey cooking instructions to a spanish speaker	15
1.2	Total amount of computation needed to train corresponding deep learning models (log scale) [Schwartz 20].	16
2.1	Illustration of Talos, a giant automaton made of bronze and forged by Hephaistos to guard the shore of Crete.	23
2.2	Illustration of Pandora, a woman molded in clay by Hephaistos at Zeus request.	23
2.3	Schematic representation of an artificial neuron.	25
2.4	Representation of a simple neural network with 2 inputs, 2 hidden layers of 4 neurons each and 2 outputs. Biases and non-linear activation functions are usually not represented for the sake of clarity.	26
2.5	The tanh activation function and its derivative.	28
2.6	The ReLU activation function and its derivative.	29
2.7	The Leaky ReLU activation function and its derivative.	30
2.8	The sigmoid activation function and its derivative.	30
2.9	The SiLU activation function and its derivative.	31
2.10	Visual representation of a simple Markov Decision Process	33
2.11	Reinforcement learning feedback loop.	34
2.12	An example of different representations for the same data	39
2.13	Usual auto-encoder architecture.	40
2.14	Usual variational auto-encoder architecture with the reparametrization trick.	42
3.1	Illustration of a classical transfer learning setup with computer vision models	46
3.2	Samples of tasks that PaLM-E can achieve [Driess 23].	47
3.3	Illustration of the domain randomization concept	48
3.4	Illustration of the Cross-Agent Transfer Learning philosophy	49
3.5	Illustration of the naive experiment setup	51
3.6	Learning curves when using direct transfer of the hidden layers	52
3.7	Auto-encoders training with an Euclidean distance between embeddings as the similarity metric[Gupta 17].	54
3.8	Robots considered for the experiments performing the proxy tasks[Gupta 17].	55
3.9	Illustrative example of a Universe with 4 different worlds (2 robots and 2 tasks) [Devin 17].	55
3.10	Training modules	56
3.11	Grid of tasks and robots considered for their experiments. As shown, one world was not seen during training and later tested on to study generalization [Devin 17].	57

3.12	Schematic representation of the UNN pipeline[Mounsif 23] and its three different modules. The red task-specific module is at the center with the robot-modules on each side	58
3.13	Schematic representation of the UNN module training with and without the Base Abstracted Modeling on a pick and place task	61
3.14	Schematic representation of some the robots used for the experiments on the UNN method. From left to right: Generic-3 robot, Berkeley Blue, Kuka-LWR, Leg Type 1, Leg Type 2. Image taken from [Mounsif 23] . . .	62
3.15	Schematic representation of a tennis-table setting, one of the tasks used for the experiments on the UNN method. Image taken from [Mounsif 23] . . .	63
4.1	Schematic representation of the Latent Space UNN pipeline	66
4.2	Illustration of the state pairing procedure for two robots with different morphologies.	67
4.3	Illustration of the bases training procedure for two robots with different morphologies.	68
4.4	Bases fitting process	70
4.5	Illustration of the UNN training procedure.	70
4.6	Latent space UNN learning process	72
4.7	Considered Universe for LS-UNN experiment	73
4.8	Kinematic diagrams of the considered robots	74
4.9	Peg insertion task setting with the physical robots.	76
4.10	Considered tasks for the experiments for LS-UNN	77
4.11	Primitive reaching task. The violet reaching target can move freely inside a reachable pre-defined working space.	78
4.12	Averaged reconstruction error (on the test set)	78
4.13	Latent space for the Braccio/Panda pair after applying PCA for dimensionality reduction and visualization on a 2D space.	79
4.14	Mean and standard deviation for each latent variable. The mean and standard deviation of the latent variables are the same across robots.	79
4.15	Neural network architecture of the full UNN network	80
4.16	Neural network architecture of the full PPO agents. We suppose a task of dimension k performed by a robot with n DoF.	80
4.17	Close-up look on the red square hole and blue peg used for the peg insertion task.	81
4.18	Pick and place task. Training curves on considered robots	85
4.19	Ball catcher task. Training curves on considered robots	86
4.20	Peg insertion task. Training curves on considered robots	87
4.21	Peg insertion task. Over-fitting performance curves on considered robots .	88
4.22	Variations based on the Sawyer robot in MuJoCo. Joints are represented by white rings and are shuffled to create different kinematic chains [Chen 18].	89
4.23	Samples of environments used for the experiments on HARL. Multiple variations of these robots were designed by changing links length.	90
4.24	Top: schematic representation of the modular robot-agnostic policy. Bottom: robots considered for the experiments. Image taken from [Ghadirzadeh 21]	91
4.25	Example of graph structure for a Walker-Ostrich	91
4.26	Experiments performed in [Zhang 21]: (a) sim2real visual adaptation, (b) cross-physic transfer and (c) cross-morphology transfer.	93
4.27	Morphologies considered for the locomotion task in [Wan 20]	94

5.1	Schematic representation of the delay aware UNN	98
5.2	Robots considered for the experiments	99
5.3	Transfers considered for the robots	100
5.4	Physical experiments setup. Left robot performs the task, while the right robot is used only to hold one end of the gutter.	101
5.5	Training curves. All the agents were trained for 400000 steps.	103
5.6	Ball trajectories with 0.3, 0.8 and 0.5 as desired ball positions.	105
5.7	RL policy (Q-Network) using a forward model to act on undelayed observations [Derman 21].	108
5.8	RL policy acting on a environment with observation delay using information-state I_2 to derive a delayed action [Nath 21].	109

List of Tables

4.1	Pick and Place task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.	82
4.2	Ball Catcher task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.	83
4.3	Peg Insertion task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.	84
4.4	Peg Insertion task: performances on the physical robots. Results are reported as percentage of task success over 28 trials.	84
5.1	Sim2sim transfer. Performances obtained for the UNN transfer on the simulated robots. Results are displayed with delay aware method on the left / delay unaware method on the right.	103
5.2	Sim2real transfer. Performances obtained for the vanilla transfer and the UNN transfer on the physical robots. Results are displayed as delay aware method on the left / delay unaware method on the right.	106
A.1	Performance obtained for UNN agents on the Pick and Place task and zero-shot transfers.	117
A.2	Performance obtained for UNN agents on the Peg Insertion task and zero-shot transfers.	118
A.3	Performance obtained for UNN agents on the Ball Catcher task and zero-shot transfers.	118

Acronyms

AI Artificial Intelligence

BAM Base Abstracted Modeling

CATL Cross-Agent Transfer Learning

CDMP Constant Delayed Markov Decision Process

CV Computer Vision

DA-UNN Delay Aware Universal Notice Network

DoF Degree of freedom

DTW Dynamic Time Warping

LS-UNN Latent Space Universal Notice Network

MDP Markov Decision Process

ML Machine Learning

MLP Multi-Layer Perceptron

NLP Natural Language Processing

PPO Proximal Policy Optimisation

RL Reinforcement Learning

Soft-DTW Soft Dynamic Time Warping

SOTA State-of-the-art

TL Transfer Learning

UNN Universal Notice Network

VAE Variational Auto-Encoder

Remerciements

This research was supported by the French Research Agency ANR through the AIM project and by the project ANITI (ANR-19-P3IA-0004)

Résumé

Malgré de nombreuses améliorations concernant l'efficacité des méthodes d'apprentissage par renforcement en robotique, l'entraînement à partir de zéro nécessite encore des millions (voire des dizaines de millions) d'interactions avec l'environnement pour converger vers un comportement performant. Dans le but d'atténuer ce besoin important de données, sans pour autant perdre en performances, une possibilité prometteuse est de se tourner vers l'apprentissage par transfert. Le but de cette thèse est d'explorer l'apprentissage par transfert dans le contexte du RL dans le but spécifique de transférer des comportements d'un robot à un autre, même en présence de divergences morphologiques ou d'espaces état-action différents. En particulier, cette thèse présente un processus de réutilisation des connaissances passées acquises par un robot (source) sur une tâche pour accélérer (ou même éviter) le processus d'apprentissage d'un robot différent (cible) sur la même tâche. La méthode proposée s'appuie d'abord sur une phase de pré-entraînement non supervisée pour apprendre un espace latent robot-agnostique à partir des trajectoires collectées sur un ensemble de robots. Ensuite, il est possible d'entraîner un modèle à l'intérieur de cet espace pour résoudre une tâche donnée afin de produire un module de tâche réutilisable par n'importe quel robot partageant cet espace de caractéristiques communes.

En outre, cette thèse s'attaque au problème du transfert simulation vers réel lors du transfert d'un comportement appris dans un simulateur, particulièrement la gestion des retards qui est peu prise en compte dans la littérature. En effet, nous montrons que les modèles qui ne tiennent pas compte des délais diminuent considérablement leurs performances lorsqu'ils sont testés sur un robot physique, où le matériel et le système sensoriel introduisent inévitablement des délais. Ainsi, l'approche développée est simple mais efficace pour entraîner des agents de manière à ce qu'ils puissent gérer une gamme de retards définie par l'utilisateur.

À travers plusieurs tâches robotiques et plateformes matérielles hétérogènes, à la fois en simulation et sur des robots physiques, cette thèse montre les avantages de ces approches en termes d'amélioration de l'efficacité d'apprentissage et de performance. Plus précisément, une généralisation instantanée est établie dans certains cas, où les performances après le transfert sont conservées. Dans le pire des cas, les performances sont récupérées après une courte adaptation sur le robot cible pour une fraction du coût d'entraînement nécessaire pour apprendre une politique avec des performances similaires à partir de zéro.

Abstract

Despite numerous improvements regarding the effectiveness of Reinforcement Learning (RL) methods in robotics, training from scratch still requires millions (or even tens of millions) of interactions with the environment to converge to a high-performance behavior. In order to alleviate this huge need for data without losing performance, one promising avenue is Transfer Learning (TL). The aim of this thesis is to explore Transfer Learning in the context of RL, with the specific aim of transferring behaviors from one robot to another, even in the presence of morphological divergences or different state-action spaces. In particular, this thesis presents a process for reusing past knowledge acquired by a robot (source) on a task to accelerate (or even avoid) the learning process of a different robot (target) on the same task. The proposed method first relies on an unsupervised pre-training phase to learn a robot-agnostic latent space from trajectories collected on a set of robots. Then, it is possible to train a model within this space to solve a given task, in order to produce a task module that can be reused by any robot sharing this common feature space.

In addition, this thesis tackles the problem of simulation-to-real-world transfer when transferring a model trained in a simulator, with a focus on delay management, which is often overlooked in the current literature. Indeed, we show that models oblivious to delay significantly drop in performance when tested on a physical robot, where the hardware and sensory system inevitably introduce delay. The approach developed is a simple but effective one for training agents to handle a user-defined range of delays.

Through several robotic tasks and heterogeneous hardware platforms, both in simulation and on physical robots, this thesis shows the benefits of these approaches in terms of improved learning efficiency and performance. More specifically, we report zero-shot generalization in some instances, where performance after transfer is preserved. In the worst case, performance is recovered after a short adaptation on the target robot for a fraction of the training cost required to learn a policy with similar performance from scratch.

Contents

1	Introduction	14
1.1	Initial discussion	14
1.2	Thesis context	16
1.2.1	Challenges	16
1.2.2	Material and scientific background	17
1.3	Contributions	18
1.3.1	Learning a robot-agnostic feature space	18
1.3.2	Dealing with multiple delay for sim2real transfer	18
1.3.3	Leveraging event-based camera for low-latency tracking	19
1.3.4	Diver gesture recognition	19
1.4	Manuscript Layout	19
1.4.1	Introduction to Machine Learning	19
1.4.2	State-of-the-art and problem statement	19
1.4.3	Transferring skills by aligning representations	20
1.4.4	Dealing with delay for simulation to reality transfers	20
1.5	Scientific publications	20
1.5.1	International	20
1.5.2	National	21
2	Preliminaries: Machine Learning	22
2.1	Introduction	22
2.2	Machine Learning	23
2.3	Neural Networks and Deep Learning	24
2.3.1	Artificial Neurons	24
2.3.2	Multi Layer Perceptrons	25
2.3.3	Activation Functions	28
2.3.4	Dealing with overfitting	30
2.4	Supervised Learning	31
2.5	(Deep) Reinforcement Learning	32
2.5.1	Motivations	32
2.5.2	Markov Decision Process	32
2.5.3	Solving MDPs with Reinforcement Learning	34
2.5.4	Reinforcement Learning Concepts	35
2.5.5	Proximal Policy Optimization	37
2.5.6	Current challenges in Reinforcement Learning	38
2.6	Representation Learning	38
2.6.1	Auto-encoders	39
2.6.2	Variational Auto-encoders	41

2.7	Conclusion	42
3	Transfer Learning in Reinforcement Learning	43
3.1	Motivations	43
3.2	Transfer Learning	44
3.2.1	Definitions	44
3.2.2	Computer Vision	45
3.2.3	Natural Language Processing	45
3.2.4	Simulation to real world	47
3.2.5	Reinforcement Learning	47
3.3	Problem statement	48
3.3.1	Formalization	48
3.3.2	A naive attempt	50
3.4	Foundational Works	52
3.4.1	Invariant Feature Space	53
3.4.2	Modular Network Policies	55
3.4.3	Universal Notice Network	57
3.5	Conclusion	62
4	Latent Space Universal Notice Network	64
4.1	Motivations	64
4.2	Towards Zero-Shot Cross-Agent Transfer Learning via Aligned Latent-Space Task-Solving	65
4.2.1	Preliminaries	65
4.2.2	Modules Training	67
4.3	Experimental setup	71
4.3.1	Considered robots	72
4.3.2	Considered tasks	72
4.3.3	Modules Training	75
4.4	Results	79
4.4.1	Zero-shot transfers	82
4.4.2	UNN fine tuning	83
4.4.3	Agent’s training	85
4.4.4	Over-fitting	87
4.5	Relation to prior works	88
4.5.1	Learning a robot-agnostic policy	88
4.5.2	Learning a correspondence mapping	92
4.5.3	Using a common feature space	94
4.6	Conclusion	95
5	Delay Aware Universal Notice Network	96
5.1	Motivations	96
5.2	DA-UNN	97
5.2.1	Constant Delayed Markov Decision Process (CDMP)	97
5.2.2	Solving a CDMDP	97
5.2.3	Delay Aware UNN	98
5.3	Experimental setup	99
5.3.1	System Architecture and Robots	99
5.3.2	Task description	100

5.3.3	Delay Aware UNN creation	101
5.4	Results	102
5.4.1	Training	102
5.4.2	Transfer	102
5.4.3	Discussion and perspectives	107
5.5	Relation to prior works	107
5.5.1	Model-based approaches	107
5.5.2	State-augmented approaches	108
5.6	Conclusion	110
A	Dealing with misaligned trajectories	115
A.1	Trajectory pairing	115
A.2	Latent trajectories alignment	115
A.3	Bases training	116
A.4	Experiments	117
A.4.1	Training setup	117
A.4.2	Results	117
A.5	Discussion and Perspectives	118

Chapter 1

Introduction

1.1 Initial discussion

The industrial revolution was largely fuelled by the many technological innovations that took place in the early 19th century. Many historians attribute this to the railway boom of 1840, made possible by the invention of the steam locomotive. What followed was a frantic race for production, aimed at ever-increasing efficiency and lower costs. It was for this purpose that Frederick Winslow Taylor developed his scientific organization of work in order to propose an optimal manufacturing method [Taylor 14]. Production is broken down into repetitive tasks performed by specialized workers. Ford then drew on this philosophy to develop the assembly line, taking the precepts of Taylorism to the extreme. Machines played a key role in this profound transformation of production techniques, initially providing simple mechanical assistance to alleviate the workers' workload. Since then, a great deal of scientific and engineering effort has gone into automating processes and tasks previously performed by humans. Naturally, mechanical and laborious tasks were the first successes of this new scientific discipline called "Control engineering". In particular, it enables the creation of fully-automated production lines, with all assembly tasks performed by robots, therefore boosting efficiency as well as eliminating hazardous and/or repetitive tasks for workers. However, it was not until the late 2000s [Goodfellow 16], with the rise of Artificial Intelligence (AI), that we saw the emergence of systems with a versatility and flexibility approaching our own. This breakthrough unlocked a wide range of applications previously thought to be achievable only by humans, such as vision [Chai 21], natural language processing [Otter 20], and decision-making [Grigorescu 20]. Multiple innovations empowered workers and developers with more sophisticated tools to carry out complex tasks more efficiently. In the current context, and in view of the lightning progresses made in this field, it is easy to imagine that robots will play an increasingly important role in our daily lives as they become more and more autonomous.

However, at the time of writing, training a model to perform even a relatively simple task requires a staggering amount of experience and testing. This is mainly due to the fact that these systems start learning without any prior knowledge of the task, the environment or their own bodies. In contrast, humans and intelligent beings, have a remarkable capacity to adapt and learn. The main reason behind this acquisition efficiency, still beyond the reach of machines, is undoubtedly the transfer of knowledge, which means they do not have to learn everything from scratch. The most fundamental and basic support for this concept is contained in the genome, refined and transmitted from generation to generation to endow newborns with a certain number of skills useful for their development

and survival. A case in point is the giraffe, which just a few hours after birth, is already able to stand upright and run to escape predators [Langman 77]. Therefore, it seems that these individuals are pre-programmed or initialized, not at random, like most AI models but according to the genetic information they have inherited. Endowing machines with similar prior knowledge to significantly speed up skill acquisition is still an open research problem.

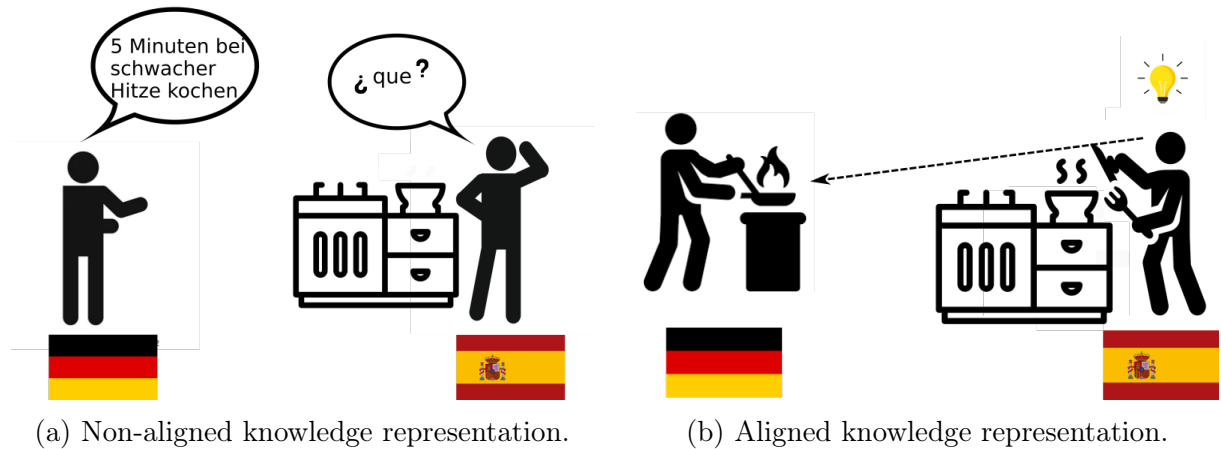


Figure 1.1: German speaker trying to convey cooking instructions to a Spanish speaker through two different mediums: (a) using language and (b) using a demonstration.

Another, more sophisticated, form of knowledge transfer widely used in the animal kingdom, consists in converting a signal (visual or audible, for example) into useful information for carrying out a task. This can be as simple as a visual demonstration, a verbal explanation or instructions. In more exotic (and as yet unexplained) cases, such as the blob (a unicellular organism), the transfer of knowledge takes place via some kind of vein that forms between the two individuals [Vogel 16]. Nevertheless, whatever the nature or medium of the information communicated, it is necessary for the expert (or source agent) and the student (or target agent) to share a *common representation* for the transfer to be useful [Sumers 20]. For example, explaining how to make a recipe in German to someone who speaks only Spanish may not achieve the desired result. However, a practical demonstration of the same instructions, either in person or on video, will provide much more information. Indeed, given the morphological similarities that exist between individuals of the same species, it is generally very easy to re-appropriate the behavior demonstrated by the expert. This applies even to relative morphological differences, such as between a child and an adult [Jones 09]. Even more impressive, it seems possible to establish correspondences with an individual of very different morphology, so as to deduce a strategy adapted to our own, thereby enabling us to achieve the same result as the one demonstrated. Once again, it is not trivial to define a shared representation in robotic domain for knowledge transfer in between AIs, particularly when morphologies differ.

In this PhD thesis, we tackle both unsolved research questions: how do we imbue prior knowledge into the learning process for efficient training? How can we define a common representation space for skill transfer? More specifically, we focus on adapting learnt behavior from one morphology to another as it is particularly relevant to robotics. Indeed, each manufacturer has its own design method, which inevitably leads to kinematic and dynamic differences in robot models. Thus, transferring a control strategy from one type of robot to another is complex, as it involves finding a common control

structure, notwithstanding their distinct and unique physical structure. Nevertheless, enabling robots to appropriate and re-use knowledge acquired by another robot is a crucial step towards reducing the training costs and energy footprint of robotic learning. A number of works already address this issue, but present practical limitations that motivated the work presented in this thesis manuscript.

1.2 Thesis context

1.2.1 Challenges

A growing number of researchers are questioning and worrying about the increasing energy requirements of State-of-the-art (SOTA) deep learning models. As previously reported in [Schwartz 20], a paper focusing on eco-friendly deep learning (Green AI), the computation required by SOTA is growing exponentially and doubling every few months, which amount to a $300.000\times$ increase in 6 years (between 2012 and 2018).

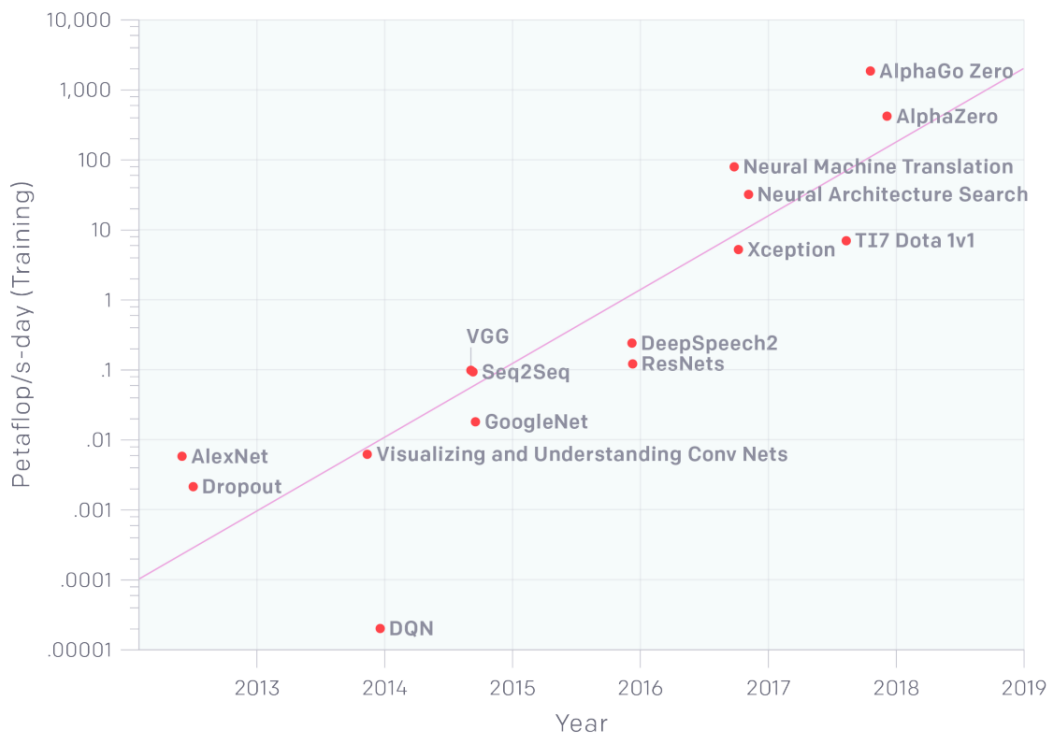


Figure 1.2: Total amount of computation needed to train corresponding deep learning models (log scale) [Schwartz 20].

At the top of the computational resources curve in Figure 1.2, we can see several Reinforcement Learning models specialized in board games playing (namely AlphaGo and AlphaZero) dominating other contemporary models. From a more robotics-flavored point of view, learning a successful gait controller requires a massive amounts of samples and computations: respectively two billion samples and 400 million samples for a quadrupedal robot in [Tsounis 20] and [Rudin 22] and approximately 1 billion samples for a humanoid robot in [Radosavovic 23] (on 4 A100 GPUs). Other robotic applications such as in-hand manipulation with a shadow-hand [Andrychowicz 20] achieves a cube re-orientation task

using an astonishing 6144 CPUs and 8 GPUs for 50 hours. As a follow-up work, the same robotic hand solved a rubiks cube manipulation task with even more compute power (29440 CPUs and 64 V100 GPUs). From a more industrial robotics standpoint, a robotic arm learned to push a puck in [Peng 18b] using 8 hours of computation with 100 cores.

The massive computations involved, not only results in a significantly substantial carbon footprint, but also causes detrimental effects on research inclusivity due to the associated expensive financial costs. Energy-efficiency of deep (reinforcement) learning models is getting increasing attention from the AI community, with some authors [Schwartz 20] advocating to use efficiency as an evaluation criterion for research alongside accuracy and related measures. Among the considered solutions for an energy-efficient training, transfer learning is one of the most popular in modern deep learning applications. It significantly reduces the amount of data and compute power needed by re-using or re-purposing knowledge previously acquired for another task or another model. It is widely used and highly effective in Computer Vision and Natural Language Processing, but only started to attract attention in the RL field recently. Indeed, only a couple of prior works proposed methodologies to mitigate the data-hungriness of SOTA Reinforcement Learning algorithms. Most notably, Mehdi Mounsif [Mounsif 20] proposed an efficient framework for TL between robots with morphological differences, but the experimental results were limited to the digital world on simulated robots. This thesis follows on from this earlier work by validating the transfer approach on multiple physical robotic arms. Additionally, we propose an novel approach to learn a robot-agnostic feature space to support the transfer of knowledge between robots, thereby avoiding the need for manual feature engineering.

The choice of using RL to learn controllers is motivated by the impressive advances made in this field. It's an easy-to-use tool, and powerful in its ability to generalize to situations not previously encountered. Exploring this control paradigm is therefore of scientific interest.

1.2.2 Material and scientific background

The research presented in this thesis was carried out at the Institut Pascal in Clermont-Ferrand, within the Image, Perception Systems and Robotics (ISPR) laboratory. More specifically, I was part of the Modeling, Autonomous and Control of Complex Systems (MACCS) team. Two of my supervisors Sébastien Lengagne and Benoit Thuilot were also part of this team. This research group mainly focuses on the modeling and control of mobile and manipulative robots, robot vision and visual control. Its research applications are closely linked to robotized manufacturing. This thesis was also carried out in collaboration with the Toulouse Systems Analysis and Architecture Laboratory (LAAS), and more specifically the Gepetto team of which my thesis director, Olivier Stasse a senior CNRS researcher, is the team leader. Olivier Stasse is also part of the Artificial and Natural Intelligence Toulouse Institute (ANITI).

The Institut Pascal laboratory is equipped with several 6 Degree of freedom (DoF) UR10 and 7 DoFs Franka Emika industrial robots which were used for our experiments. For all our transfer experiments, these robots are assumed to have the same physical capabilities even if their mechanical structures differ. Additionally, we also had access to low-cost quality robots such as a 5 DoFs Braccio robot, used in some experiments. From a software point-of-view, all computations were run on a consumer-grade GPU (GeForce RTX 3080) and a 12 core AMD Ryzen 9 CPU. We leveraged the Pytorch deep learning library for all our deep learning based experiments. The simulator used throughout this

thesis was Unity 3D [Juliani 18], a free physic engine. As a result, the equipment used is within the reach of any laboratory, allowing reproducibility and re-use of our contributions. Furthermore, this low energy-consumption setup is aligned with the Green AI philosophy mentioned previously. We also release our source code (links available in the next section), mostly written in Python and C#. Although other and more powerful simulators have recently been proposed such as Isaac gym [Makoviychuk 21a], the delay between their public release and the end of the thesis did not allow us to make the transition. This simulator is proving more efficient at parallelizing experience gathering and training on a massive scale, thus saving time. However, this innovation is complementary to our contribution, as our approach reduces the amount of experience required, further reducing training time.

1.3 Contributions

1.3.1 Learning a robot-agnostic feature space

As mentioned earlier, this PhD thesis aims to make substantial contributions to the field of Transfer Learning in Robotics. Its main focus is directed towards transferring skills from one robot to another, regardless of their respective physical structure. Our approach relies heavily on the creation of a shared and aligned representation between the agents considered. The definition of this common space is primordial for downstream transfers as it relates how one task could be performed similarly by two different robots. We make use of an unsupervised training procedure to discover the shared structure from recorded demonstration of a basic task. This circumvents the need to manually define the shared feature space unlike prior works. As a result, our method is more flexible and does not require domain-specific knowledge. We demonstrate that our learned latent space is highly suited for transfer by training a re-usable task module within it, which generalizes (instantly in some cases) to unseen morphologies. Additionally, we also show that the bias induced by the demonstrations used to craft the shared feature space can ease the learning process. The source code for this contribution can be found at <https://github.com/sabeaussan/LS-UNN>.

1.3.2 Dealing with multiple delay for sim2real transfer

Our second contribution (but first in chronological order) deals with transfer from simulation to reality. More specifically, we consider the relatively frequent case where the agent is trained on a simulator with instant execution of its actions and immediate access to the state of the environment, but next deployed on hardware with imperfect actuators and delayed perceptions. To overcome this obstacle, we introduce a simple training method to deal with a broad range of delays, thereby preserving the transferability of agents trained in simulation to robots with different morphologies and intrinsic delays. By bridging the gap between simulation and reality, we further enable the seamless application of learned skills in practical robotic scenarios. The source code for this contribution can be found at <https://github.com/sabeaussan/DelayAwareUNN>.

1.3.3 Leveraging event-based camera for low-latency tracking

Other related works are still in progress at the time of writing. In particular, experiments conducted jointly with a trainee aimed at transferring a highly dynamic ball-catching task for which our transfer approach has been validated in simulation, to a real platform. The additional contribution compared to DA-UNN is the use of an event-based camera instead of an RGB frame-based camera for the low-latency acquisition of the position of the ball.

1.3.4 Diver gesture recognition

On a different note, a collaborative work with a PhD student from the University of Toulon, Bilal Ghader, is also underway and as such not discussed in this manuscript. The aim of this joint effort is to enable the recognition of underwater diving gestures by training a classifier from gestures only made in the air domain. Indeed, water data are very expensive to collect in contrast to air data, and this scarcity hinder performance of the resulting diving gesture recognition model. Training a model with air data is therefore appealing from a practical standpoint. However, there is significant differences between the execution of the same gesture in the air and in the water, which prevent the classifier trained with abundant air data to generalize well to water data. This challenge can be addressed using a similar approach as our first contribution. More specifically, our proposed solution relies on the creation of a shared representation space for air and water data in order to train a domain agnostic classifier.

1.4 Manuscript Layout

This manuscript is organized as follows:

1.4.1 Introduction to Machine Learning

This dissertation begins with a brief introduction to Artificial Intelligence and Machine Learning (ML) in chapter 2 to make our contributions easier to understand. It starts with a description of the Multi-Layer Perceptron (MLP) architecture, widely used throughout our work. Then discuss the nuts and bolt of this type of models, and provide the underlying theoretical mechanisms of neural networks. Following this short introduction to deep learning, we review the currently prevailing paradigms of machine learning starting with Supervised Learning to introduce some general key concepts. Next, we move on to Reinforcement Learning, the main ingredient in our contribution. We discuss some of its current limitations and describe Proximal Policy Optimisation (PPO), the learning algorithm used in this report. Finally, Representation Learning is introduced through the Auto-Encoder framework, an essential component of our proposed transfer method.

1.4.2 State-of-the-art and problem statement

Chapter 3 centers around the main problematic of this thesis: transfer learning, delving into its purpose within the field of machine learning and its potential benefits for Reinforcement Learning. Following this, it formally defines the general transfer learning problem and showcases various examples of its application across a range of ML sub-fields. Then, we zoom in on the specific setting we are ultimately interested in: Cross-Agent

Transfer Learning (CATL), a branch of TL applied to Reinforcement Learning specifically for agents with different physical structure. We show through a simple experiment that a naive TL approach based on the current paradigm widely spread in other ML fields is inefficient, thereby requiring more sophisticated methods. Consequently, the chapter proceeds to examine three CATL approaches and their limitations, which motivated our research endeavor.

1.4.3 Transferring skills by aligning representations

Having clearly defined the problem and the shortcomings of some early solutions, Chapter 4 features our core contribution: a simple, yet efficient approach to learn a shared and aligned feature space to support the transfer of knowledge between agents. By leveraging the Universal Notice Network (UNN), a CATL framework developed by Mehdi Mounsi during his PhD, we show through several tasks and robots that skills can be transferred with instant generalization to other robots morphologies without having to explicitly define the shared representation unlike prior works. Our method is a two step process, first requiring to match similar states across robots to learn a shared and abstract representation space. The next step involves learning to solve the task inside the aforementioned space, thus removing the dependency between the task-solving strategy and the agent’s morphology.

1.4.4 Dealing with delay for simulation to reality transfers

The last chapter of this PhD thesis addresses the issue of sim2real adaptation, which involves transferring models trained in simulators to real-world environments. More specifically, we highlight the impact of delays on model performance when transitioning from simulation to physical robots, an often overlooked factor in the adaptation process. Consequently, this chapter discusses a simple practical solution to train RL agents able to robustly deal with a user-defined range of delays, making them well-suited for transfer across robotic platforms. We demonstrate the validity of our approach by means of a dynamic task where delay management is critical, across two robot morphologies and three different delays.

1.5 Scientific publications

Our first two contributions have been the subject of one or more scientific publications. The first contribution has been submitted to a journal (Journal of Artificial Intelligence Research) on April 17 of 2023, but is still under review. Our second contribution has been published in two conference, International Conference on Intelligent Robots and Systems (IROS) in 2021 and Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA) in 2022. Our work on the event-based camera and the diving gesture classifier are still under development, and will be the subject of further publication once finalized.

1.5.1 International

Journal

- **Beaussant, S.**, Lengagne, S., Thuilot, B., Stasse, O. (2023). Towards Zero-Shot Cross-Agent Transfer Learning via Aligned Latent-Space Task-Solving. Journal of

Conference

- **Beaussant, S.**, Lengagne, S., Thuilot, B., Stasse, O. (2021, September). 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).

1.5.2 National

Conference

- **Beaussant, S.**, Lengagne, S., Thuilot, B., Stasse, O. Delay Aware Universal Notice Network: Real world multi-robot transfer learning. In 2022 Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA)

Chapter 2

Preliminaries: Machine Learning

The primary aim of this chapter is to present the theoretical foundations that form the backbone of this PhD thesis. We begin with a concise and general overview of the Machine Learning field, which serves as our fundamental building block. We then describe the family of parametric models employed in all of our experiments in the section dedicated to deep neural networks. Finally, we delve into the various learning paradigms used throughout this manuscript, taking the opportunity to highlight their strengths and weaknesses at the same time.

2.1 Introduction

Artificial Intelligence (AI) is a discipline in computer science and engineering that focuses on creating intelligent machines which can perform tasks that typically require human intelligence. This includes reasoning, perception, natural language processing, and decision-making amongst others. This idea of endowing mindless machines with human-like cognitive abilities has been around for a long time and even precedes the birth of computers. Indeed, it can be traced back to at least the ancient Greece, with early conceptual prototypes found in mythical figures such as Talos and Pandora (see Figure 2.1 and Figure 2.2). One particularly interesting finding from these decades of AI research, is the fact that tasks that are easy for humans, such as perception and motor control, are often difficult for machines. Reciprocally, difficult tasks for average humans, such as complex mathematical calculations, are often easily solvable for computers. This paradox named after its author Hans Moravec in 1988 [Moravec 88], indicates that high-level reasoning is much easier to reproduce and simulate by a computer program than human sensorimotor skills. Hence, computer scientists struggled for many years to solve even simple vision tasks such as face recognition but succeeded to defeat chess world champions with deep blue [Campbell 02] in 1997. Indeed, games like chess have well-defined rules and a limited number of possible moves, making it possible to develop strategies that can be executed efficiently by computers, even with brute force search. Identifying faces, in contrast, involves not only recognizing facial features but also distinguishing between different individuals, regardless of subtle variations in lighting, pose, and other factors. As such, designing an expert system that incorporates hard-coded rules to perform a task that can be trivially accomplished by the human mind, has proven to be a rather difficult and inefficient approach. In fact, all these early AI algorithms assumed to be the best direction to artificial general intelligence, such as the General Problem Solver or the A^* algorithm [Russell 10], were missing one fundamental component: the ability to learn.



Figure 2.1: Illustration of Talos, a giant automaton made of bronze and forged by Hephaistos to guard the shore of Crete.



Figure 2.2: Illustration of Pandora, a woman molded in clay by Hephaistos at Zeus request.

2.2 Machine Learning

Instead of hard-coding knowledge within an AI system using manually crafted rules and inference engines, Machine Learning (ML), a subfield of AI, adapts the concept of learning to suits computers such that they can improve their performance on tasks through experience. This is typically achieved by training algorithms on large datasets and using statistical techniques and models to identify patterns in the data to make predictions. More formally, let's consider a set of *observations* $X = (x_1, x_2, \dots, x_n)$ (e.g images, sensor readings, text...), the goal of a Machine Learning algorithm is to make a prediction $\hat{Y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n)$ given the data X and a model $f(x; \theta)$ such that

$$\hat{y}_i = f(x_i; \theta), i = 1, \dots, n \quad (2.1)$$

where θ denotes the set of parameters characterizing the model f . In this context, training f means finding the θ parameters that best approximate f^* , the input/output mapping we are interested in. For instance, the optimal mapping f^* could relate an input image to its corresponding label (e.g dog or cat). Many choices of models f are available when considering Machine Learning to solve a task. Examples of such models include Gaussian Processes [Deisenroth 13], Linear Classifier [Cortes 95] and Gaussian Mixture Models [Zong 18] etc... with their respective performance depending on the task setting. However, in its contemporary incarnation, Machine Learning relies extensively on a special kind of function approximator known as Artificial Neural Networks. Over the past few decades, machine learning algorithms have proven their unreasonable effectiveness on many complex tasks including image segmentation [Kirillov 23], text-to-image generation [Rombach 22], protein folding prediction [Jumper 21] and Natural Language processing [Brown 20] to name just a few. Some products implementing these ML algorithms are today even regarded as potential societal threats due to their ability to outperform humans in a growing number of professions [OpenAI 23]. Besides their performance, ML data-driven approaches are appealing for their ability to autonomously

solve tasks and generalize beyond the data they were trained on [Neyshabur 17]. Particularly in robotics, learned controllers have unlocked a multitude of impressive results [Tsounis 20, Akkaya 19, Jangir 20, Chebotar 19] in recent years, competing with or even outperforming traditional robotic approaches. Once again, this echoes the Moravec Paradox as learning such controllers is sometimes easier than manually defining them. All of these massive achievements make Machine Learning a very promising and compelling research field worth exploring.

Due to the over-representation of Artificial Neural Networks in Machine Learning, but also in our works, the upcoming section focuses on the concepts and mathematical details under-pining this type of models. Following this, we review three primary learning paradigms composing ML: Supervised Learning, Representation Learning and Reinforcement Learning. All of these ML fields play a significant role in our contributions. Therefore, in the following section, we take the time to thoroughly explore their unique characteristics and areas of application.

2.3 Neural Networks and Deep Learning

Artificial (deep) Neural Networks (ANN) have been used to robustly solve a vast range of problems due to their expressive power and ability to approximate complex, non-linear mappings. In fact, from a theoretical perspective, ANNs can approximate any function, a result commonly known as the universal approximation theorem [Hornik 89]. Many of the most significant breakthroughs of AI can be attributed to the use of large ANNs, including Alphafold [Jumper 21], AlexNet [Krizhevsky 17], Generative Adversarial Networks (GANs) [Goodfellow 20] and many others. This section provides the theoretical background underlying modern neural networks architecture, beginning with the basic structure of a single neuron cell and progressing towards the complexity of deep neural networks.

2.3.1 Artificial Neurons

Artificial neurons are the elementary computation units of neural networks. They were designed to mimic the behavior of biological neurons. Each neuron has several input connections that receive signals from other neurons or external sources, and a single output connection that transmits its own signal to other neurons or output devices. The input signals are weighted according to the importance of each input to the neuron's function, and the neuron applies a non-linear activation function to the weighted sum of its inputs to produce its output. More formally, a neuron with weights $w \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$ receives an input $X \in \mathbb{R}^d$ and computes an output $a \in \mathbb{R}$ through the following processing steps (see Figure 2.3 for a visual representation):

$$z = w^T x + b = \sum_i^d w_i \cdot x_i + b \quad (2.2)$$

This weighted sum is then "activated" to give the final output

$$a = g(z) \quad (2.3)$$

where g is a non-linear activation function. These special kinds of functions are of the utmost importance in deep learning. Therefore, we briefly describe and discuss some of the most popular ones in section 2.3.3.

On its own, a simple artificial neuron has very limited modeling capacities. However, it becomes extremely powerful and flexible when duplicated and interconnected to create stacks of layers. This arrangement of neurons takes the form of a network, which explains its common denomination: neural network. Various types of neural networks exist, depending on their architecture and connection patterns. In this thesis, we mainly focus on the Multi-Layer Perceptron (MLP) since the majority of our research relies on it.

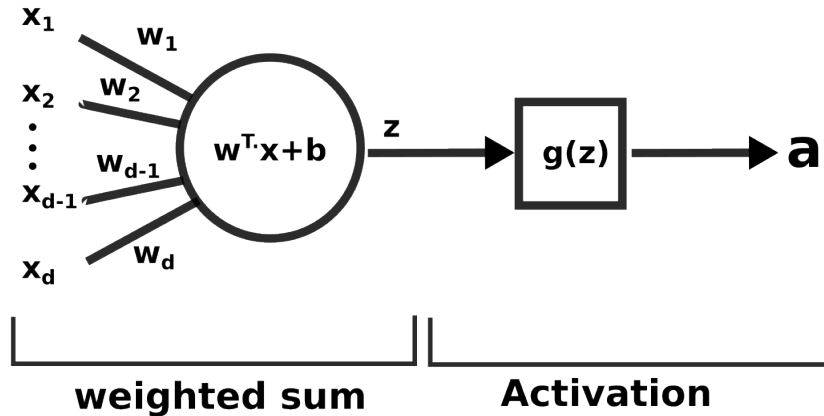


Figure 2.3: Schematic representation of an artificial neuron.

2.3.2 Multi Layer Perceptrons

This is the simplest ANN architecture but also the most versatile one. In a Multi-Layer Perceptron, each neuron in a layer is connected to every neuron in adjacent layers, but not to neurons in the same layer (see Figure 2.4). There are 3 kinds of layers:

- Input layer: it represents the input to the neural network. Strictly speaking, it does not contain neurons, simply the data in vector format fully connected to the next layer.
- Hidden layers: they are used to process the input data, producing intermediate representations at each stage which will feed the next layer.
- Output layer: this is the last layer in the network. Its objective is to perform the prediction or decision based on the final transformed representation of the data given by the upstream hidden layers. Depending on the task, this output could be a vector of joints command for a robot or probabilities for a classification problem for instance.

In deep learning, ANNs are made of several hidden layers to increase their modeling power (at least more than one). Empirically, the deeper the network is (i.e., the more hidden layers it has), the better the performance will be [He 16, Szegedy 15]. An MLP is

a feedforward neural network, meaning that the information flows in one direction, from input to output without loops or feedback connections. The successive computations and transformations of the data are defined by the weights of the neural network. Therefore, these parameters must be carefully adjusted so that the model performs the desired mapping. In practice, fitting a neural network (i.e training) is an iterative process composed of two steps. First, we need to compute the prediction of the network given an input, this is the forward pass. Then, given this output, we can compute the prediction error and adjust (i.e learn) the weights such that the error decreases over time. This process is known as the backward pass. The following subsections provide the mathematical formalization of both steps.

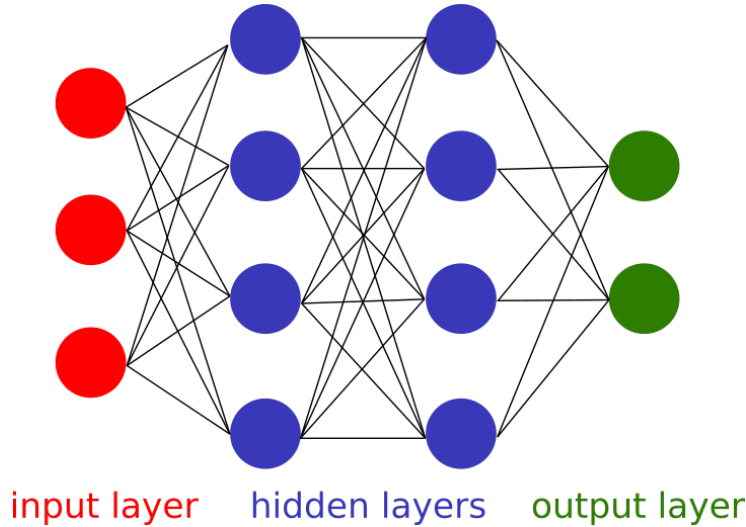


Figure 2.4: Representation of a simple neural network with 2 inputs, 2 hidden layers of 4 neurons each and 2 outputs. Biases and non-linear activation functions are usually not represented for the sake of clarity.

Forward Pass: making a prediction

During the forward pass, the input data are successively processed by each layer of the network until they reach the output layer. Let's first consider a single layer l with N neurons. The function computed by l is

$$a^{(l)} = g^{(l)}(W^{(l)} \cdot a^{(l-1)} + b^{(l)}) \quad (2.4)$$

where $W^{(l)} \in \mathbb{R}^{N \times D}$ is the weight matrix of the neurons in layer l with D the number of neurons on the previous layer, $b^{(l)} \in \mathbb{R}^N$ is the bias vector, $a^{(l-1)} \in \mathbb{R}^D$ is the activation vector of the previous layer $l - 1$ and $a^{(l)} \in \mathbb{R}^N$ is the activation vector outputted by l . Given that a neural network is often composed of multiple layers, this computation is repeated L times, with L the number of layers. As such, the full forward pass for an MLP can be written as

$$\hat{y} = (a^{(L)} \circ a^{(L-1)} \circ \dots \circ a^{(2)} \circ a^{(1)})(x) \quad (2.5)$$

Equation (2.5) states that the predicted output \hat{y} of the network is obtained by composing the outputs of all the layers up to the final output layer L given x , the input vector.

Backward Pass: computing the gradient

Once the forward pass is completed, we can derive its error with respect to a pre-defined objective function or loss function. It is common practice to use stochastic gradient descent (ascent) to minimize (maximize) the optimization criterion and update the weights of the neural network. As such, it is necessary to compute the gradient of the loss with respect to the parameters of the ANN. This procedure is commonly known as the "backward pass" as it involves computing the gradient starting from the output layer and propagating it backwards through the network until reaching the first layer. Given a loss E , we can compute the gradient of E with respect to any weight in the network using the chain rule. Starting from the last layer L :

$$\frac{\partial E}{\partial w_{ij}^{(L)}} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} \quad (2.6)$$

for weight $w_{i,j}^{(L)}$ of the last layer L . To alleviate and improve the readability of the back-propagation equations, we define by $\delta_j^{(l)}$ the quantity iteratively computed and propagated backward as:

$$\delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}} \quad (2.7)$$

Given this new notation, equation (2.6) can be re-written in a more compact form

$$\frac{\partial E}{\partial w_{ij}^{(L)}} = \delta_j^{(L)} \cdot \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} \quad (2.8)$$

For the previous layer $L - 1$, the method is the same, except that we make use of some of the computations done at the previous step represented by $\delta_j^{(L)}$

$$\frac{\partial E}{\partial w_{ij}^{(L-1)}} = \delta_j^{(L)} \cdot \frac{\partial z_j^{(L)}}{\partial a_j^{(L-1)}} \frac{\partial a_j^{(L-1)}}{\partial z_j^{(L-1)}} \frac{\partial z_j^{(L-1)}}{\partial w_{ij}^{(L-1)}} = \delta_j^{(L-1)} \cdot \frac{\partial z_j^{(L-1)}}{\partial w_{ij}^{(L-1)}} \quad (2.9)$$

Finally we can recursively compute the gradient with respect to any weight $w_{ij}^{(l)}$ in the network, for an arbitrary layer l with

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_j^{(l+1)} \cdot \frac{\partial z_j^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} \quad (2.10)$$

where $\delta_j^{(l+1)}$ has been obtained when computing the gradient for layer $l + 1$. Once the full gradient has been computed, it can be used to update the neural network in the direction that decreases the prediction error with gradient descent:

$$\theta_{k+1} = \theta_k - \nabla_{\theta} E \quad (2.11)$$

where θ are the weights of the network in vector form, and k is the training iteration.

2.3.3 Activation Functions

The effectiveness of contemporary artificial neural networks in modeling complex relationships can be largely attributed to the use of non-linear activation functions. In fact, if a deep neural network exclusively employs linear activation functions, it will be restricted to only approximate linear mappings, irrespective of the number of stacked layers [Minsky 69]. Therefore, in order to achieve universal approximation capabilities, it is essential to incorporate non-linear activation functions. Besides the non-linearity prerequisite, activation functions should also have some extra properties to ease learning:

- Continuously differentiable to allow for gradient computation and backpropagation.
- Non-saturating to provide a robust learning signal. An activation function f is saturating if:

$$\lim_{|v| \rightarrow \infty} |\nabla f(v)| = 0 \quad (2.12)$$

The back-propagation algorithm involves multiplying gradients with the chain rule, so small gradients would eventually shrink the gradient product towards 0. In practice, this prevents the first layers of the neural network from learning. This issue is commonly known as the *vanishing gradient*.

We introduce below the activation functions used in this work along with their derivative:

The Hyperbolic Tangent:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \quad (2.13)$$

$$\tanh'(x) = (1 - \tanh(x)^2) \quad (2.14)$$

The Tanh activation function was popular several decades ago, but it is now only occasionally used in Reinforcement Learning since it saturate and may cause vanishing gradients (see Figure 2.5). Nevertheless, Tanh remains useful as an output activation function since it compresses its input values within the range of $[-1, 1]$.

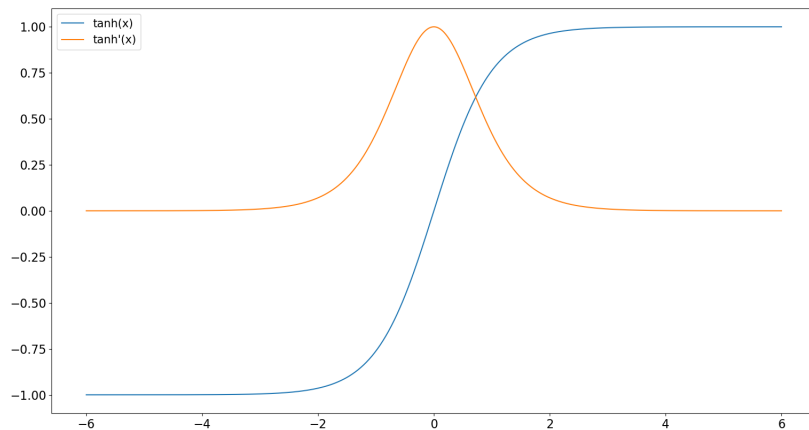


Figure 2.5: The tanh activation function and its derivative.

Rectified Linear Unit:

$$\text{ReLU}(x) = \max(0, x) \quad (2.15)$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \quad (2.16)$$

The ReLU function [Fukushima 69] is almost default in modern deep learning. It was designed to prevent the vanishing gradient issue, resulting in more stable and faster training for deep neural networks [Glorot 11]. It is also very efficient to compute as it involves simple operations. However, it suffers from its own specific problem: dying neurons. This problem appears when the neuron pre-activations are always negative. As a result, the neuron becomes essentially inactive and outputs zero for almost all inputs. In this state, the neurons are not contributing to the prediction and therefore do not receive any gradient to learn, remaining indefinitely inactive. In worst case scenarios, a large portions of the neurons in the network can "die", resulting in a reduction of the model's capacity. Fortunately, this problem can be mitigated by appropriately initializing the model [He 15].

Leaky ReLU:

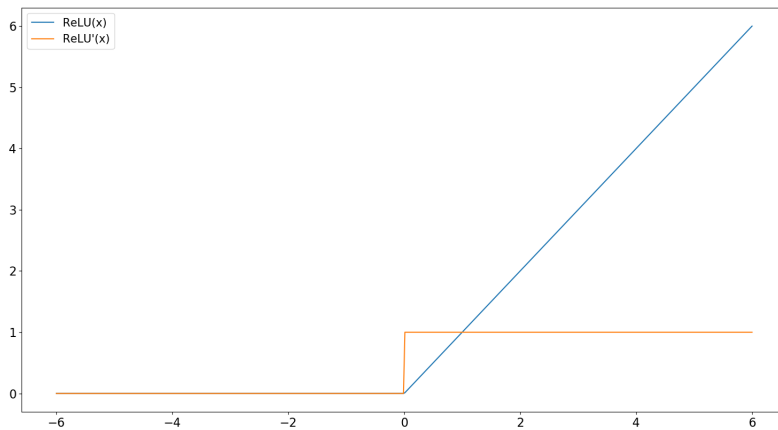


Figure 2.6: The ReLU activation function and its derivative.

$$\text{Leaky ReLU}(x) = \max(0.01x, x) \quad (2.17)$$

$$\text{Leaky ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0.01 & \text{else} \end{cases} \quad (2.18)$$

Alternatively, one may turn to the Leaky ReLU [Maas 13] activation function to avoid the dying neurons problem altogether. It allows a small positive gradient to flow when its input is negative.

Sigmoid:

$$\sigma(x) = \frac{1}{e^{-x} + 1} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad (2.19)$$

The sigmoid activation function (often denoted σ) is very similar to the Tanh function. As such, it suffers from the same vanishing gradient issue. Its range is $[0; 1]$ which makes it convenient to obtain values analogous to probabilities. For this reason, the sigmoid function is still used as an activation function for the output layer.

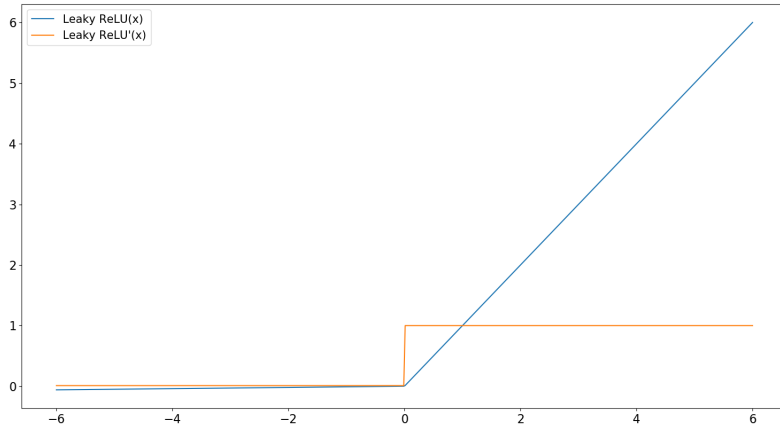


Figure 2.7: The Leaky ReLU activation function and its derivative.

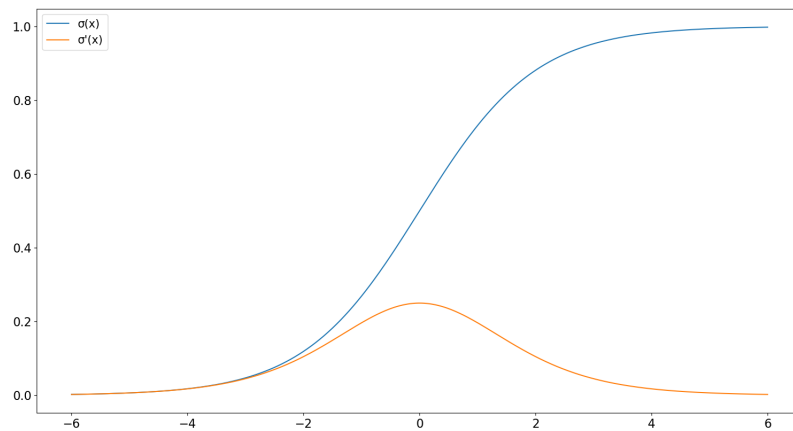


Figure 2.8: The sigmoid activation function and its derivative.

Sigmoid Linear Unit (or Swish):

$$\text{SiLU}(x) = x \cdot \sigma(x) \quad \text{SiLU}'(x) = x \cdot \sigma'(x) + \sigma(x) \quad (2.20)$$

The SiLU [Ramachandran 17] is based on the sigmoid activation function but does not inherit from the same pitfalls. Indeed, the SiLU function looks like a smooth ReLU and has been shown to outperform it in Reinforcement Learning settings [Elfwing 18]. Along with the Leaky ReLU it is one of the main alternatives to the regular ReLU activation function.

2.3.4 Dealing with overfitting

Given the flexibility and modeling capacity of artificial neural networks, it is very common to run into an *overfitting* issue. Overfitting occurs when a neural network is too complex and starts to fit the training data too closely, leading to poor performance on unseen data. In a sense, it is simply memorizing the data rather than finding the statistical patterns underlying them. There are two main ways to prevent this problem:

- Increasing the amount of data, with additional fresh data points or creating more variations by altering the available data with random transformations (e.g random rotations or random crops for images). This is known as data augmentation

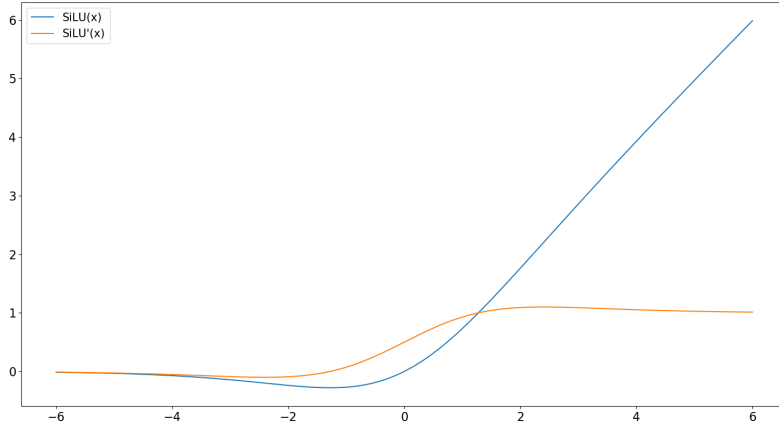


Figure 2.9: The SiLU activation function and its derivative.

- The other approach consists in "crippling" the network in order to limit its modeling capacity. This could be achieved with regularization by adding a penalty term to the loss function in order to discourage large weights or with dropout layers [Hinton 12, Srivastava 14] to randomly drop out some of the neurons during training, forcing the network to learn more robust features.

In essence, overfitting is an issue occurring in machine learning, when the model is asked to make predictions on data coming from a distribution that differs from the one used to train the model. As such it is very frequently encountered in Transfer Learning.

2.4 Supervised Learning

Supervised Learning (SL) is one of the most common and widely used paradigms in Machine Learning. It is powerful and flexible enough to be leveraged in a variety of real-world applications such as digits recognition [LeCun 89], autonomous vehicle driving [Bojarski 16] or medical diagnosis [Gupta 18]. SL involves learning from a labeled dataset, where each input data is paired with the corresponding output or target values. Using the associated ground-truth labels, it is possible to teach the model about the expected answer given the presented data. This is analogous to the kind of learning students experience in schools where the "right" solution is provided by the teacher for a given problem. Hence the *supervised* learning denomination. Nevertheless, the purpose of Supervised Learning is not to make the model memorize the right answers, but rather understand the mapping between the input and output data. Hopefully, the trained model will then be able to accurately predict the outputs for new, unseen but related input data.

The supervised learning process requires a labeled dataset $D = \{(x_i, y_i)\}_{i=1:N}$ of length N , where $x_i \in \mathbb{R}^K$ designates the input data of dimension K and $y_i \in \mathbb{R}$ the corresponding label or ground-truth. The model f , expected to approximate a desired mapping f^* , is queried during training to predict $\hat{y}_i = f(x_i)$, its current best guess for y_i . Then, the error between the prediction and the expected output is computed using a *loss* function denoted C

$$e_i = C(\hat{y}_i, y_i) \quad (2.21)$$

The overall optimized loss L of the model is then computed using the entire dataset

following

$$L = \frac{1}{N} \sum_i^N e_i \quad (2.22)$$

Finally, f is adjusted such that its error L is iteratively reduced during training (with gradient descent for instance).

2.5 (Deep) Reinforcement Learning

2.5.1 Motivations

Let us now consider a sequential decision-making problem and suppose that we want to learn the optimal decision strategy. As a concrete example, we can think about the game of chess. Leveraging Supervised Learning to train a model at this complex game requires a comprehensive dataset containing examples of probable situations with their corresponding optimal decisions. For many interesting applications however, collecting an "optimal behavior" dataset could be infeasible as the combinatorial size of the state-action space would be prohibitively large (around 10^{11} possible states in a chess game for instance). Furthermore, devising such a dataset requires unlimited access to an expert in order to associate each possible state with the corresponding best action. As a consequence, the model would at best reach an expert-level performance, which might still be highly sub-optimal for very complex tasks. For all these reasons, SL is not an efficient paradigm to solve sequential decision-making problems. Fortunately, Reinforcement Learning (RL) [Sutton 18] focuses on learning a desired behavior. Unlike SL approaches, the model does not require access to a labeled dataset of examples for the optimal action in a given setting. Rather, the agent interacts with its environment to collect its own experience and learns by trial-and-error. As such, it is particularly suited for tasks that require making sequences of decisions over an arbitrarily long horizon of time. For many robotic applications, it is a convenient tool that yields successful control strategies without requiring an analytical model of the dynamics to be defined, an often tedious task. Together with Reinforcement Learning, ANNs bootstrapped the Deep Reinforcement Learning field, leading to groundbreaking results in robotics [Akkaya 19], animation [Peng 18a], game-theory [Silver 18, (FAIR)† 22]. It is even considered in critical tasks where failure is not an option such as fusion control [Degraeve 22].

2.5.2 Markov Decision Process

Most of the time, sequential decision-making problems are framed as a Markov Decision Process (MDP). This mathematical framework provides a convenient way to model the interaction between an environment where outcomes are sometimes partly random and a decision maker, often called an *agent* in the RL context. The agent can be regarded as a bodiless entity containing the logic required to make decisions. In contrast, the concept of environment can be blurry in some cases, but the common definition states that it comprises everything, including the agent's embodiment. For instance, the motors, sensors and mechanical parts of a robot should be viewed as constituents of the environment, rather than components of the agent. In its most general and abstract formulation, a MDP is a discrete-time stochastic control process typically defined by a 5-tuple $MDP = \langle S, A, P, R, \rho_0 \rangle$. A toy example is provided in Figure 2.10.

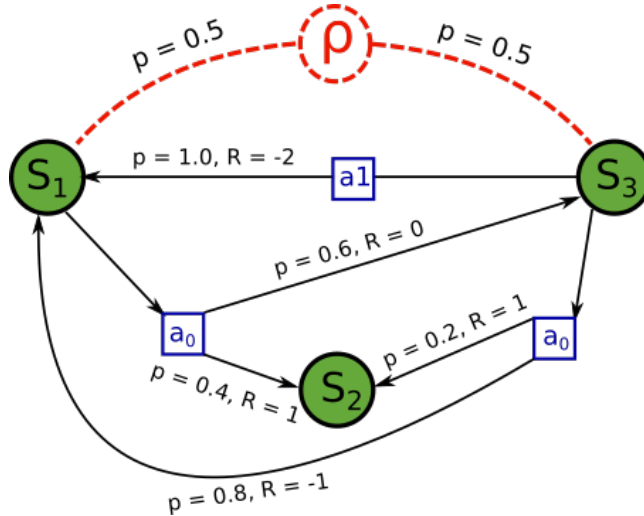


Figure 2.10: Visual representation of a simple Markov Decision Process. States are in green and actions in blue. The initial state distribution ρ is depicted in red and assigns equal probabilities to start state 1 and 3.

State space S

S is the collection of all valid states available in the environment. An observation $s_t \in S$ thus represents the state of the environment at time-step t . The initial state distribution of the MDP is denoted by ρ_0 . Depending on the task setting, the state s_t is a complete or partial description of the world. In the latter case, the MDP formulation is slightly changed to define a Partially Observable MDP (POMDP). This special type of MDPs is often encountered when the observations are noisy, delayed or incomplete (e.g RGB images only provide information about what is currently viewed). The state space can be either continuous (infinite) or discrete (finite) or a mixture of both, with an arbitrary number of dimensions. For the rest of this manuscript, unless specified otherwise, we assume a fully observable MDP with a continuous state space.

Action space A and policies

A is the set of all feasible actions the decision-maker can make. The specific choice of action a_t in state s_t is implemented by the agent's policy denoted by π . A policy can be described as a mapping from perceived environmental states to the actions that should be undertaken when in those states. To put it simply, it defines the agent's behavior. The decision-making process can be either deterministic or probabilistic. In the first case, the stochastic policy is a probability distribution over actions a conditioned on the state s :

$$a \sim \pi(\cdot|s) \tag{2.23}$$

In practice, stochastic policies are more suitable for large state-action spaces because they promote exploration. Otherwise, π is deterministic:

$$a = \pi(s) \tag{2.24}$$

In all of our experiments, we used stochastic policies with a continuous action space.

The environment dynamic P

At each time-step, the environment evolves and gives rise to a new state following its transition probability distribution.

$$s_{t+1} \sim P(\cdot | s_t, a_t) \quad (2.25)$$

Any MDP satisfies the Markov property which means that only the present state s_t and the current agent's action a_t condition the future.

The reward function R

The concept of reward function is of major importance as it defines the optimal behavior. Concretely, it is the mapping $R : S \times A \times S \mapsto \mathbb{R}$ giving feedback r_t to the decision-maker after transitioning from state s_t to state s_{t+1} , in response to its action a_t . As such, it is the learning signal that guides the agent through its learning process. Defining a reward function that would generate the desired behavior is not a trivial task [Amodei 16]. The main design choice is whether the reward should be sparse or dense. In the first case, the reward signal is not very informative with a positive feedback only in a restricted part of the space (e.g when it reaches the end goal) and negative/zero otherwise. Dense means that the reward function is shaped to guide the agent at each state towards its goal (with a reward inversely proportional to the distance to target for instance). In this thesis, we decided to work with dense rewards as they often results in more stable training and performance.

2.5.3 Solving MDPs with Reinforcement Learning

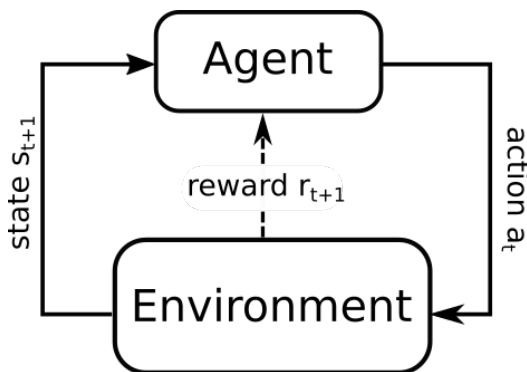


Figure 2.11: Reinforcement learning feedback loop.

In a Markov Decision Process, the goal of the agent is to find a policy that maximizes over a horizon T its cumulative discounted reward (or return) $G(\tau)$ defined as:

$$G(\tau) = \sum_{t=0}^T \gamma^t r_t \quad (2.26)$$

where $\gamma \in [0; 1]$ is a hyper-parameter weighting distant rewards and τ is a trajectory, that is a sequence of states and actions

$$\tau = ((s_0, a_0), (s_1, a_1), (s_2, a_2), \dots) \quad (2.27)$$

Thus, we wish to find an optimal policy π^* such that:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{\tau \sim \pi} [G(\tau)] \quad (2.28)$$

where \mathbb{E} denotes the expectation operator. Reinforcement Learning is a formidable tool for finding the policy π^* that maximizes the expected sum of rewards. Typically, RL is a closed-loop process as the learning system's actions influence its later inputs. The usual flow of events is the following (depicted in Figure 2.11), at each time-step t and for the duration of an episode of arbitrary length T :

1. the agent observes the current state of the environment $s_t \in S$
2. it draws an action $a_t \in A$ from its policy π
3. the environment evolves using its transition probability distribution P
4. and the agent is provided with $s_{t+1} \sim P(\cdot | s_t, a_t)$ and a feedback signal $r_t = R(s_t, a_t, s_{t+1})$

2.5.4 Reinforcement Learning Concepts

When considering the choice of a RL algorithm, it is important to wonder what we want to learn and how. Indeed, RL encompasses a broad taxonomy of learning algorithms, each requiring access to different models. A common branching point is whether we should opt for a model-free or a model-based approach. Model-based RL often involves learning or using a model of the environment, i.e. $P(s_{t+1} | s_t, a_t)$ the transition probability distribution. This often results in a more sample efficient training over methods that are model-free. However, for complex environments, learning an accurate model is a very tedious task. Model-free RL, on the other hand, requires a lot of samples to converge but results in a better-performing policy. Additionally, another fundamental design choice is whether we want to directly optimize the policy and/or learn value functions (defined in the following subsection). In practice however, most modern RL algorithms use both concepts to maximize performance. For instance, Proximal Policy Optimization [Schulman 17], a model-free Reinforcement Learning algorithm, guides the optimization process of the policy using value-functions. Below, we review all the components required to implement PPO, the algorithm used in all our experiments and described in Section 2.5.5.

Value Functions

There is no denying that "intuition" frequently plays a significant role in the decision-making process of intelligent beings. People tend to follow their instinct when they are faced with a decision regarding a distant and unpredictable future. This concept of "intuition" could be interpreted as an estimate of "how good is likely to be the state that I will end in, if I make this decision". Roughly speaking, it involves weighting future outcomes by their respective likelihood of occurrence. A similar idea can also be translated to Reinforcement Learning. Unsurprisingly, the notion of "goodness", in the context of RL, is defined in terms of the expected sum of rewards the agent can get following its policy π . As such, value functions are usually defined with respect to a specific policy. Two kinds of value functions are available:

- **The state-value function:** Denoted by $V^\pi(s)$, it is the expected return when starting in state s and following π thereafter. It is mathematically defined as

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | S_t = s \right] \quad (2.29)$$

- **The action-value function:** Similarly, we can define the value of taking action a in state s under a policy π . Denoted by $Q^\pi(s, a)$, it is the expected return of starting in state s , choosing action a and following π ever after. It is formally expressed as

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | S_t = s, A_t = a \right] \quad (2.30)$$

Advantage Function

The state-value function and the action-value function are both valuable tools in many scenarios. However, in some cases, it is better to evaluate the relative superiority of an action compared to others on average, rather than in absolute terms (as measured by Q values). Formally, the advantage function is defined as the difference between the state-value function and the action-value function for a given state and action. Mathematically, it can be expressed as:

$$A^\pi(s, a) = V^\pi(s) - Q^\pi(s, a) \quad (2.31)$$

This quantity is used in several reinforcement learning algorithms to update the policy of the agent as discussed in Section 2.5.5.

Policy Gradient

As a recall, the objective of a Reinforcement Learning algorithm is to solve Equation (2.28). Let π_θ be a stochastic policy parameterized by θ and consider:

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}[G(\tau)] \quad (2.32)$$

A straightforward method to find the optimal policy π_θ^* would be to apply gradient ascent iteratively using

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \quad (2.33)$$

where α is the step size. The gradient of expected return $J(\pi_\theta)$ (also interpreted as policy performance), $\nabla_\theta J(\pi_\theta)$, has a special name: the policy gradient. Algorithms that use this gradient to optimize the policy are referred to as policy gradient methods. This gradient can be estimated in a model-based or model-free fashion. For model-free methods, it was shown [Sutton 99] that the policy gradient $\nabla_\theta J(\pi_\theta)$ can be written as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) G(\tau) \right] \quad (2.34)$$

This formula has a nice interpretation: taking a gradient step will update the probabilities of trajectories under π_θ in proportion to $G(\tau)$, the return. The above expression can be estimated by sample mean with

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \left(\sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) G(\tau) \quad (2.35)$$

where D is a buffer containing all the collected trajectories prior to policy update. This formulation provides a closed-form expression to compute the gradient. It is the basis of many subsequent model-free algorithms which improved its formulation to address some of its weaknesses.

2.5.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [Schulman 17] is a model-free reinforcement learning algorithm that uses policy gradients to update the policy. It is designed to prevent catastrophic performance decreases that can occur when the steps taken to update the policy with the vanilla policy gradient are too large. The algorithm achieves this by taking extra care during policy updates, resulting in improved stability and sample efficiency. PPO is proven to provide monotonic improvements, and it has achieved state-of-the-art results on multiple challenging benchmarks with continuous state-action spaces. It draws inspiration from trust-region optimization methods [Schulman 15], which enforce an additional constraint on the optimization process to ensure that the new policy is relatively close to the old one.

PPO only uses first-order derivatives of the objective function, avoiding the need to compute the expensive Hessian matrix off the constraint unlike prior trust-region based methods. This is achieved using a simple clipping function with the following objective:

$$L(s, a, \theta_{old}, \theta) = \min(r(\theta)A^{\pi_{\theta_{old}}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)A^{\pi_{\theta_{old}}}(s, a)) \quad (2.36)$$

with

$$r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} \quad (2.37)$$

where θ refers to the parameters of the policy and *clip* is a mathematical operation that restricts a value within a specific range. It takes an input value and limits it to a minimum and maximum value, ensuring that the output remains within the defined boundaries.

$$\text{clip}(x, a, b) = \begin{cases} a, & \text{if } x < a \\ x, & \text{if } a \leq \text{value} \leq b \\ b, & \text{if } x > b \end{cases}$$

PPO has the particularity of performing multiple epochs of stochastic gradient ascent on the same training buffer before recollecting fresh on-policy data. The policy parameters at the beginning of each epoch are thus denoted by θ_{old} . Essentially, this objective function ensures that the policy ratio $r_{\theta}(a|s)$ stays in the $[1 - \epsilon, 1 + \epsilon]$ range to provide conservative policy updates. The hyper-parameter ϵ roughly says how far away the new policy is allowed to go from the old one at each gradient step. In practice however, we do not have access to the advantage function $A^{\pi_{\theta_{old}}}$ so we need to estimate it. We use the Generalized Advantage Estimation (GAE) algorithm [Schulman 16] to do so, as it offers an extra hyper-parameter λ to trade-off between the bias and the variance of our advantage estimator. Mathematically, it is defined as

$$\hat{A}_t^{GAE(\gamma, \lambda)}(s, a) = \sum_{l=0}^{\infty} (\gamma \lambda)^l (r_{t+l} + \gamma V^{\pi_{\theta}}(s_{t+l+1}) - V^{\pi_{\theta}}(s_{t+l})) \quad (2.38)$$

where $V^{\pi_{\theta}}$ is the state-value function. As shown, this expression only requires $V^{\pi_{\theta}}$ to be computed. In our case, this quantity is approximated by another neural network often

called the *critic*. It is trained by stochastic gradient descent to minimize

$$L_{critic}(s_t) = (V_{critic}(s_t) - G_t)^2 \quad (2.39)$$

thus converging to the true expected return. We used PPO to train the models discussed in this report since it currently achieves state-of-the-art results on multiple domain, including robotic.

2.5.6 Current challenges in Reinforcement Learning

Despite noticeable achievements, Reinforcement Learning is still a highly active research topic. Modern RL algorithms are struggling in several aspects of the learning process.

Exploration

One of the main challenges faced in Reinforcement Learning is the proper exploration of the state-action space which is magnified for high-dimensional environment. In practice, insufficient exploration potentially leads to suboptimal policies if the agent did not happen to come across the highly valuable, but extremely unlikely states during training. This is similar to the issue of optimizing non-convex objective functions where the optimization process can get stuck into a local minimum. Exploration is also of primary importance when dealing with sparse reward environment, as the outcome of a particular action may not be visible until thousands more decisions have been made. One simple method to improve exploration is to add noise to the network parameters [Lillicrap 16] or adding an entropy bonus term in the reward function [Harojoja 18, Ahmed 19]. More sophisticated algorithms provide the agent with a reward signal that is independent of the external task, encouraging exploration of the environment for its own sake [Eysenbach 19, Pathak 17, Tang 17, Bellemare 16].

Reward shaping

Shaping a reward (i.e defining its analytic form) to obtain the desired behavior also is challenging in its own right. Indeed, it is difficult to predict how the agent will end up acting under a given reward function [Amodei 16]. Misspecified or ill-defined reward function are a common issue for practitioners. For instance, a reward function solely motivating a bipedal agent to cover more ground when learning to walk is likely to yield a highly suboptimal gait, such as crawling on the ground, exhibiting erratic body movements or dragging one of its feet. One family of solutions is to recover the reward function given demonstrations of a near-optimal behavior (e.g motion capture data of a human walking) [Ng 00, Ziebart 10, Finn 16] and then perform policy optimization given the obtained reward function, a process known as Inverse RL. Another successful kind of approaches, dubbed imitation learning, directly optimizes a policy such that it minimize the divergence between the demonstrations and the roll-outs sampled from the policy [Ho 16, Schaal 96, Peng 18a].

2.6 Representation Learning

In the early days of machine learning, the features upon which the classification algorithms operated had to be hand-designed, a process called feature-engineering [Goodfellow 16].

At the time, the quality of the features used was paramount to the algorithms success (see Figure 2.12 for an illustration).

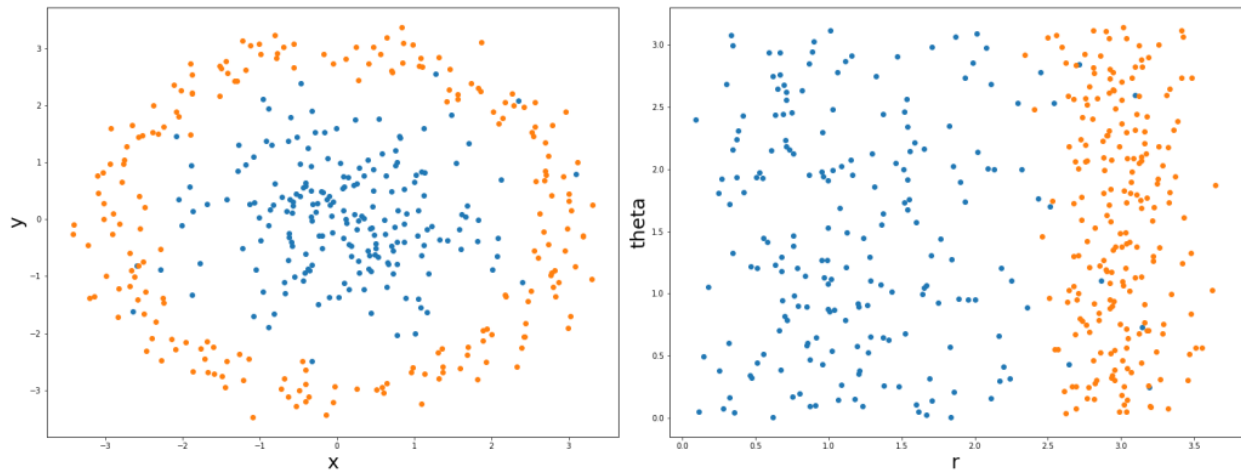


Figure 2.12: An example of different representations for the same data: suppose we want to classify two categories of data with a linear model. In the plot on the left, the data are represented using Cartesian coordinates. The data are not linearly separable meaning that the task is impossible. In the plot on the right, we represent the same data with polar coordinates and the task becomes simple to solve.

With the growing interest and applicability of deep neural networks, researchers discovered that it is possible to learn representations and that it often result in much better performance than what could be obtained with hand-designed features [Goodfellow 16]. This approach, in contrast to feature engineering, is called *representation learning* [Bengio 13]. Despite being a rather old sub-field of Machine Learning, representation learning is still a very active and relevant topic. Its goal is to learn representations of the data that make it easier to extract useful information when building classifiers or other predictors. Some of its most prominent families of algorithms are the auto-encoders and Principal Component Analysis (PCA) [Bro 14] which both provides an unsupervised procedure to extract useful features from the data. However, PCA is limited in its modeling capacity as it only uses a linear projection to represent the data. Nevertheless, it is still very useful as a dimensionality-reduction tool to analyze and visualize high-dimensional datasets. We will make use of PCA to visualize our learned feature space later in subsection 4.2.2. Instead, in this section we focus our discussion on the auto-encoder family which can be seen as a non-linear equivalent to PCA. In particular, we describe one of its most prominent representative, widely used in our work, the variational auto-encoder.

2.6.1 Auto-encoders

Auto-encoders [Kramer 91] span an entire class of specific models that undergo training to approximate an identity mapping. In other words, given an input $x \in X$, the model must predict $\hat{x} = x$. Although, at first glance, this task may appear trivial and useless, what makes it truly interesting is the architecture of the neural network used. Indeed, they implement the concept of information bottleneck where the data are first compressed

into a lower-dimensional space, before being reconstructed. The main goal is to get rid of the extra redundant or low entropy information that may add unnecessary complexity to the data. As a consequence, we obtain a compact and meaningful representation of our dataset which, hopefully, only captures its main factors of variation. More specifically, an auto-encoder is composed of two parts, each with a predefined role (see Figure 2.13):

- an encoder f_ϕ , parametrized by weights ϕ , which takes an input data point $x \in X$ and maps it to a lower-dimensional representation $z \in Z$. This new space Z is often referred to as the latent space.

$$z = f_\phi(x) \tag{2.40}$$

- and a decoder g_θ , parametrized by weights θ , which ensures that the encoded information z is sufficient to reconstruct x . This encourages the network to discover a compressed but meaningful representation of the data.

$$\hat{x} = g_\theta(z) \tag{2.41}$$

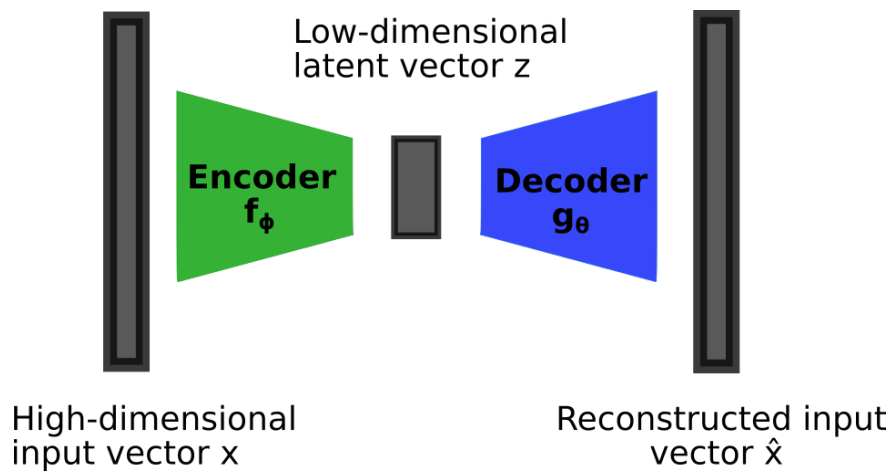


Figure 2.13: Usual auto-encoder architecture.

The idea that high-dimensional data can be expressed in a simpler and more compact space is known as the manifold hypothesis [Fefferman 16]. As such, auto-encoders can be seen as a way of learning the data manifold. The objective function used to train this types of models simply minimizes the euclidean distance between the initial input and its reconstructed counterpart. More formally:

$$L(\phi, \theta) = \|x - \hat{x}\|^2 = \|x - g_\theta \circ f_\phi(x)\|^2 \tag{2.42}$$

From a practical point of view, training an auto-encoder requires to find a trade-off between complexity and accuracy. On the one hand, one should always strive to have the smallest possible latent space to prevent the network from encoding irrelevant information about the dataset or over-fitting by memorizing it. But on the other hand, reconstruction accuracy should be high to ensure that the information bottleneck contains all the useful information.

The auto-encoder framework has given birth to a broad taxonomy of diverse use cases, from denoising input signal [Vincent 10] to anomaly detection [Zhou 17]. Beyond the simple representation learning aspect of these models, it is also possible to add extra regularization terms to the loss function in order to give desirable properties to the generated

latent space. This is typically done to address some of the limitations of traditional auto-encoders. More specifically, if we simply try to generate new data points by interpolating between points in the latent space, or randomly sampling latent vectors, the decoded result is likely to be random and meaningless. This is due to a form of over-fitting from the decoder, not motivated during training to give a meaning to points outside of the training data distribution. As a result, auto-encoders cannot be used as generative models as they lack proper regularization of their latent space.

2.6.2 Variational Auto-encoders

Among the many declinations of the auto-encoder formulation, one of the most celebrated and widely used by practitioners is the Variational Auto-Encoder (VAE) [Kingma 14]. It enhances the regular auto-encoder framework by regularizing its latent space such that it can be used for data generation. The main difference lies in the probabilistic approach of the VAE in which the decoder and the encoder are both stochastic rather than deterministic. Instead of mapping the input into a fixed latent vector, we map it into a probability distribution. As such, the probabilistic encoder defines a conditional distribution $f_\phi(z|x)$, which approximates $p(z|x)$, the true data posterior that relates x to a corresponding distribution over z . Most of the time this distribution is assumed to be a simple conditional Gaussian:

$$f_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x)^2 I) \quad (2.43)$$

As shown, both the mean μ_ϕ and standard deviation σ_ϕ are parametrized by ϕ . In practice, the probabilistic encoder is made of a single neural network with two heads which outputs the mean vector μ_ϕ and a vector of standard deviations σ_ϕ . Using these two quantities, it is then possible to sample a latent vector $z \sim f_\phi(z|x)$. In contrast, regular auto-encoders deterministically project the processed inputs to the latent space. This probabilistic approach is an effective procedure to train the decoder to generalize to latent areas rather than specific encodings as it is exposed to wider variations. Similarly, the generative part is handled by the probabilistic decoder $g_\theta(x|z)$ which remaps a given $z \in Z$ to its corresponding $x \in X$ by means of a neural network with weights θ . However, since the decoder is trained to maximize the log-likelihood of the input data, it approximates the data generative process and is able to create unseen samples. Simply maximizing the log-likelihood loss (i.e minimizing the reconstruction loss as previously), is not sufficient for data generation. Indeed, this unconstrained optimization procedure will still likely result in a scattered latent space as μ_ϕ and σ_ϕ are free to take any real values to help the reconstruction. As a result, the decoder is still likely to over-fit to some areas in the latent space, leaving in-between portions without meaningful decodings. Moreover, there is no easy way to sample from it as it lacks structure. Solving above issues requires an extra regularizing term to the training loss:

$$L(\phi, \theta) = -\mathbb{E}_{z \sim f_\phi(\cdot|x)}[\log g_\theta(x|z)] + \beta D_{KL}(f_\phi(z|x) || \mathcal{N}(0, I)) \quad (2.44)$$

where $D_{KL}(f_\phi(z|x) || \mathcal{N}(0, I))$ is the Kullback-Leibler divergence which quantifies the distance between $f_\phi(z|x)$ the approximated posterior and the prior $\mathcal{N}(0, I)$. More precisely, for two continuous probability distribution p and q :

$$D_{KL}(p||q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx \quad (2.45)$$

The β hyper-parameter introduced in [Higgins 17] is used to trade off between reconstruction and regularization. Minimizing this alternative loss function causes the latent space to have some interesting and desirable properties:

- It encourages the latent space to be distributed as a target (or prior) distribution. Knowing a priori the structure of the latent space is necessary if we want to be able to sample new data. In our case the prior is simply $p(z) \sim \mathcal{N}(0, I)$ which is easy to sample from and tractable to compute.
- Constraining the possible values of μ_ϕ and σ_ϕ to be normally distributed also ensures that the latent space is continuous (two close points in the latent space give two similar outputs when decoded) and complete (a point sampled from the latent space should produce an output that makes sense).

Finally, as the training objective (2.44) involves back-propagating through the stochastic process $z \sim f_\phi(\cdot|x)$, it is necessary to reparametrize sampling to compute the gradient as:

$$\begin{aligned} z &= \mu_\phi + \epsilon \odot \sigma_\phi \\ \epsilon &\sim \mathcal{N}(0, I) \end{aligned} \tag{2.46}$$

This technique is called the "reparametrization trick" and it allows for an end-to-end training of the VAE using equation (2.44).

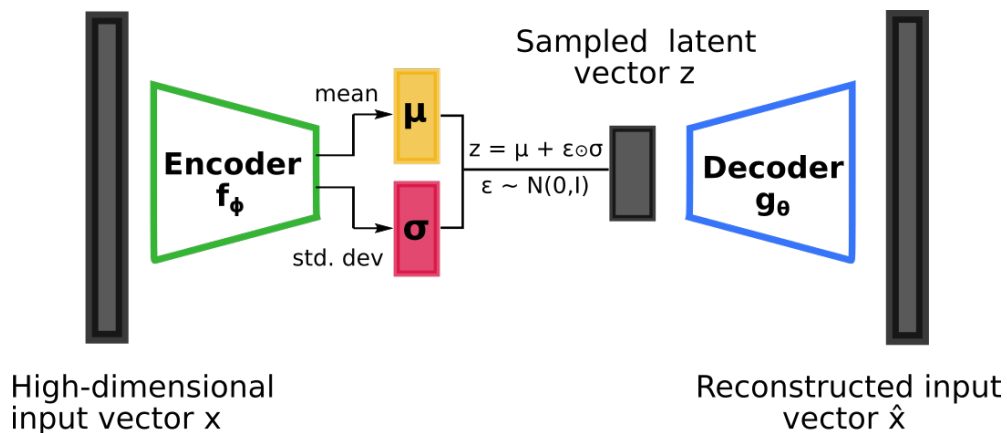


Figure 2.14: Usual variational auto-encoder architecture with the reparametrization trick.

2.7 Conclusion

In this chapter, we reviewed all the required components to fully understand the rest of this report and our contributions. We started with a mathematical description of neural networks, a particularly powerful function approximator. In the subsequent sections we also presented the Variational AutoEncoder, an important representation learning algorithm for automatic features discovery. Most importantly, we introduced the Reinforcement Learning field along with some of its shortcomings. However, we purposely left off its arguably most prominent limitation: the notorious sample inefficiency. This topic will be extensively discussed throughout this PhD dissertation, starting with its introduction in the following chapter. Additionally, the next chapter will delve into the main subject, beginning with an exploration of the motivations that justify the need for our research endeavor.

Chapter 3

Transfer Learning in Reinforcement Learning

3.1 Motivations

Ever since Deep Reinforcement Learning came to light thanks to the famous works on Deep Q-Networks by [Mnih 13, Mnih 15], multiple general and versatile RL solvers have been proposed to improve performance of RL agents. They demonstrated impressive achievements on a vast range of robotic tasks such as in-hand manipulation for rubik’s cube solving [Akkaya 19], control of quadrupedal gaits [Tsounis 20], cloth manipulation [Jangir 20] or swing-peg-in-hole manipulation [Chebotar 19]. However, people tend to overlook the massive cost required to train these models when they consider the impressive results achieved. Indeed, despite great achievements, RL methods still suffer from low sample efficiency, which means that a large amount of interactions with the environment (at least in the order of millions) is needed to obtain a high-performance policy [Da Silva 19]. For instance, training a robot hand to solve a rubik’s cube, or a quadrupedal to navigate difficult terrain, can take several days of training on some simulators, which can amount to years in realtime. One of the main explanations behind such a high computational cost, is the fact that most agents are trained from scratch, without any pre-training or prior knowledge of the task/environment. As a consequence, each time the agent is required to learn a task, it has to first discover how to articulate its body appropriately and avoid undesirable joint configurations, before being able to act optimally with respect to a reward function. Conversely, humans and certain animals possess an inherent ability to utilize past experience to efficiently learn new tasks. They can even learn from other individuals despite differences in morphology, demonstrating the ability to comprehend the similarities and disparities between their role model and themselves, and adapt their actions according to their own capabilities. Endowing robots with similar prowess would result in a significantly more sample efficient learning process, and prove useful in many situations. Practical use cases may include for instance:

- Changing an old and worn out robot on an assembly line with a new but different one. As they ultimately achieve the same task, we can transfer the knowledge from the old to the new robot to speed up the replacement process.
- Learn and prototype a task on an unreliable but cheap robot before transferring it to an industrial-grade but more expensive robot. This way we prevent potential hazard due to the inherent unpredictability of RL training on the target robot.

The primary aim of this chapter is to introduce *Transfer Learning* (TL), a general framework for leveraging past knowledge. Additionally, it formalizes the problem of Cross-Agent Transfer Learning (CATL), a specific sub-field of TL which focuses on experience sharing between agents with differing morphologies. We will examine previous efforts on CATL, which served as a basis for our own proposed method discussed in the next chapter.

3.2 Transfer Learning

To address the issue of sample inefficiency and high computational cost in RL, we direct our focus to Transfer Learning (TL). The central idea behind TL is to make use of knowledge gained from one task to improve performance on another related task. As on numerous other occasions, this concept first emerged in psychology before appearing in Machine Learning. Indeed, the theory of generalization, proposed by psychologist Charles Judd as early as 1908 [Judd 08], states that transfer from task A to task B is possible if by achieving task A the learner discovers common or general features which generalize, in part or completely, to task B . In essence, it consists in discovering the shared structure to a number of related but distinct situations. For instance, learning tennis involves principles and skills that are closely related to table-tennis or squash. Hence, an athlete going from one to another is likely to learn more efficiently than an athlete starting from scratch. It is important to note that transfer does not necessarily lead to a positive impact on learning and can even be detrimental. For instance, learning a language that has a specific grammatical structure can lead to biases and habits that a learner must overcome in order to acquire a new language with a fundamentally different structure, a problem known as linguistic interference [Lennon 08].

3.2.1 Definitions

In the ML context, the learner is represented by a model. Several prior works [Zhuang 20, Pan 10] proposed a general and formal description of TL. The concepts of domain and task must be defined in the first place:

Definition 1: (Domain) A domain \mathcal{D} is composed of two components: a feature space \mathcal{X} and a marginal probability distribution $P(X)$ such that $X = \{x_1, x_2, \dots, x_n\} \in \mathcal{X}$. Hence, $\mathcal{D} = \{\mathcal{X}, P(X)\}$.

The joint space of a robot could be regarded as \mathcal{X} , with X a particular joint configuration.

Definition 2: (Task) Given a domain \mathcal{D} , a task \mathcal{T} can be defined by a tuple $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$, where \mathcal{Y} is the label space and $f(\cdot)$ is the decision function, expected to be learned from the training data contained in \mathcal{D} . From a RL standpoint, \mathcal{Y} represents the optimal actions we want the policy $f(\cdot)$ to predict.

Finally, we can formalize the concept of Transfer Learning using both definitions:

Definition 3: (Transfer Learning) Given a source domain \mathcal{D}_S and learning task \mathcal{T}_S , a target domain \mathcal{D}_T and learning task \mathcal{T}_T , transfer learning focuses on improving the learning of the target decision function $f_T(\cdot)$ in \mathcal{D}_T using the knowledge in \mathcal{D}_S and \mathcal{T}_S , even if $\mathcal{D}_S \neq \mathcal{D}_T$.

An ideal scenario for many ML algorithm, is to have access to a large annotated training set belonging to the same domain \mathcal{D} as the test set. This would increase the odd of generalization to unseen data once deployed. However, for many interesting real world applications, this situation is rarely encountered. Collecting and manually labeling a sufficient amount of data for a task of interest, can be prohibitively expensive. As a result, it is almost the default procedure to use a deep neural network pre-trained on a large, diverse and reliable data-set. This allows the model to learn useful and reusable features. Subsequently, this pre-trained model can be used as a starting point for a new model, fine-tuned on a smaller, more specific collected data-set. In addition to improving sample-efficiency, Transfer Learning also reduces the burden of data collection and annotation for supervised learning tasks. Depending on the type of transfer, we can expect two outcomes if the transfer is successful:

- Zero-shot generalization: the pre-trained model instantly generalizes to the new domain/task without additional training required.
- Few-shot generalization: the pre-trained model generalizes over the new task/domain after some fine-tuning, i.e additional training with new data points.

For our particular setting, above definitions will be adapted in the section 3.3 dedicated to Cross-Agent Transfer Learning . To better contextualize our line of work and how it compares to modern TL approaches, the next subsections present some of the most widely used applications of TL in Machine Learning, including Computer Vision (CV) and Natural Language Processing (NLP).

3.2.2 Computer Vision

Transfer learning initially gained popularity and wide spread adoption for Computer Vision tasks. In practice, the model used for transfer is pre-trained on a large scale classification task on the imageNet data-set. With over 10 million images spanning 1000 classes, a model trained on this data-set can learn to extract a broad range of features from images, which means that the learned representation is likely to generalize to other domains [Donahue 14, Sharif Razavian 14]. Assuming that the model is a deep neural network, the first layers (called the feature extractor in this case), learn to hierarchically decompose an image into edges, shapes and contours. As these features are quite general, the feature extractor can be efficiently re-purposed for other tasks such as detection [Redmon 16, Girshick 15] and image segmentation [Kirillov 23, He 17]. After transfer of the feature extractor, all that remains to be done, is to fine-tune the final layers of the network, significantly speeding up learning of the downstream task. This simple process is illustrated in Figure 3.1.

3.2.3 Natural Language Processing

The same approach is also increasingly seen in Natural Language Processing with Bidirectional Encoder Representations from Transformers [Devlin 18] (BERT) frequently acting as a pre-trained model. This process is similar to the one used in Computer Vision, but the data-set utilized for pre-training is a massive text corpora. This allows the model to capture important information about language syntax, semantics, and context. Once

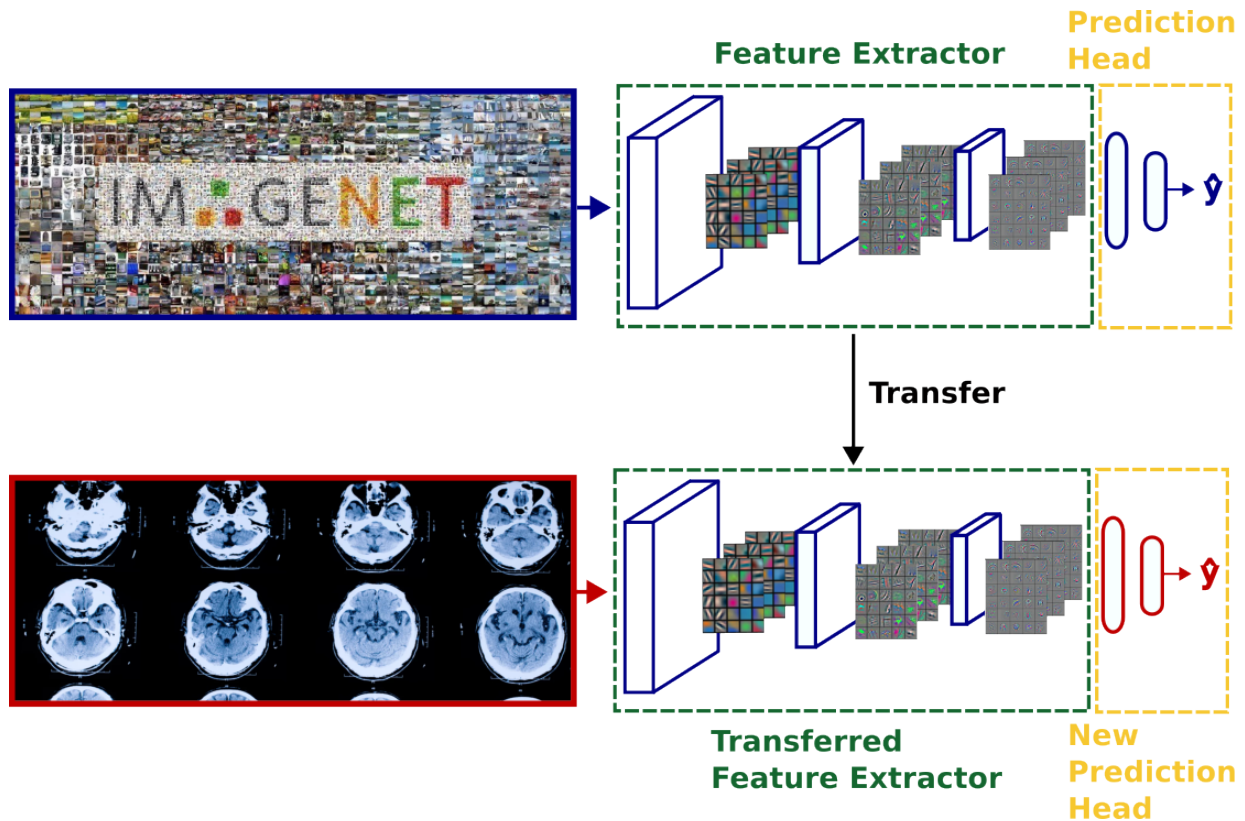


Figure 3.1: Illustration of a classical transfer learning setup with computer vision models. One model was trained on imageNet, a very large image database and acquired general feature detection capabilities. As such, the model extractor can be re-purposed and transferred to another model for a different application (medical for instance).

again, the feature extractor in BERT can be repurposed for more specific tasks, such as aspect-based sentiment analysis [Hoang 19] and question-answering [Devlin 18], with only the task-specific final layers requiring adjustment. A concurrent but similar approach, the Generative Pre-trained Transformer (GPT) [Brown 20], also relies on unsupervised pre-training on large corpus of text before fine-tuning on task-specific data-sets. As fine-tuning these large models on task-specific data-sets can be quite expensive, [Hu 22] proposed to instead train from scratch a significantly smaller model that contains ΔW , the low-rank adjusted weights of the large model. The most widely-used pre-training method in NLP involves an unlabeled dataset with a simple training objective of predicting the next word based on the given context [Devlin 18]. Despite its apparent simplicity, this pre-training method results in a powerful and robust knowledge representation. For instance, PaLM-E [Driess 23], an embodied large language model (562 billions parameters) is able to perform long-horizon planning tasks given visual and language inputs such as "bring me the chips", on 2 different robots using the same model (see Figure 3.2). Additionally, its sophisticated embedding space enables zero-shot generalization to new tasks and appropriate response to unseen directives. As usual, this model was pre-trained on a large corpus of text [Chowdhery 22] before being fine-tuned on real world data.

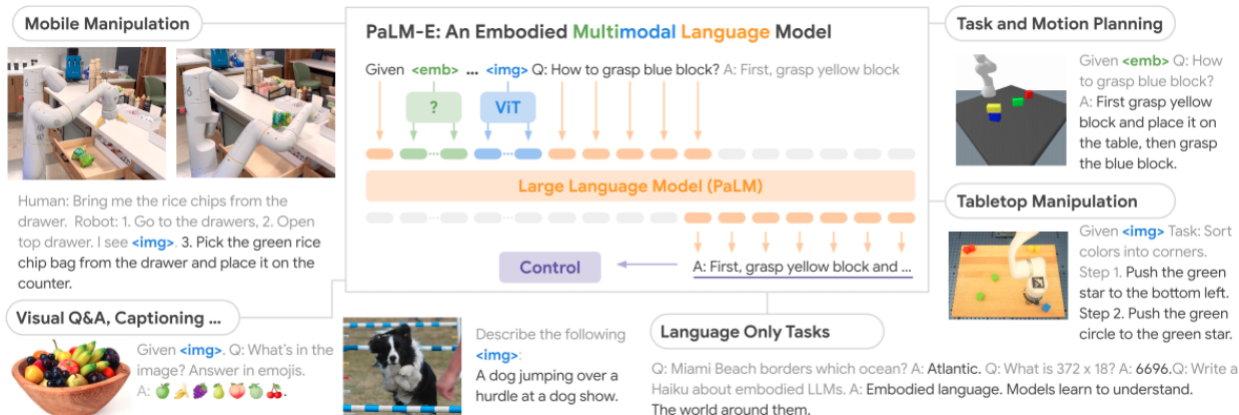


Figure 3.2: Samples of tasks that PaLM-E can achieve [Driess 23].

3.2.4 Simulation to real world

For most robotic applications, we are ultimately interested in deploying the agent on a real system. However, the exploration required by RL agents which raises safety concerns and the massive amount of training samples needed, make learning directly on the physical robot extremely hard. As such, training usually happens in a simulator to benefit from the availability of virtually unlimited data and to avoid potential physical damages. Unfortunately, reinforcement learning agents have a strong tendency to overfit to their environment. In practice, this implies that even small changes in the transition dynamic can lead to catastrophic decrease in performance. Even high-end simulators [Todorov 12, Makoviyuchuk 21b] are not able to fully replicate the richness of the real world, leading to discrepancies when transferring models from simulation to real world, known as the sim2real gap. As such, transfer from simulation (source domain) to reality (target domain) can also be regarded as a Transfer Learning problem. To mitigate the harmful effect of sim2real transfers on models performance, [Akkaya 19, Peng 18c] randomize a number of physical and dynamic parameters to expose the agent to a wide variety of possible environments (see Figure 3.3 for an illustration). In practice, the agent can adapt to a distribution of environments, hopefully encompassing the real one. Subsequent work [Chebotar 19] estimates the best mean and variance for the distributions of physical parameters from real world experience rather than manual tuning, in an expectation-maximization fashion. Mutual Alignment Transfer Learning [Wulfmeier 17] guides and accelerates the training of an agent on an actual robot by adding a reward term such that it visits states similar to its better performing virtual counterpart.

3.2.5 Reinforcement Learning

In both CV and NLP, transfer learning is straightforward to implement and often results in state-of-the-art results. However, finding a practical and general approach for pre-training and transfer learning in the context of RL policies, is still an open research problem. The RL community has yet to find and adopt a foundation model [Bommasani 21] which will provide an efficient starting point for downstream applications. One major challenge is the fact that contrary to neural networks in CV, there is no spontaneous knowledge segmen-

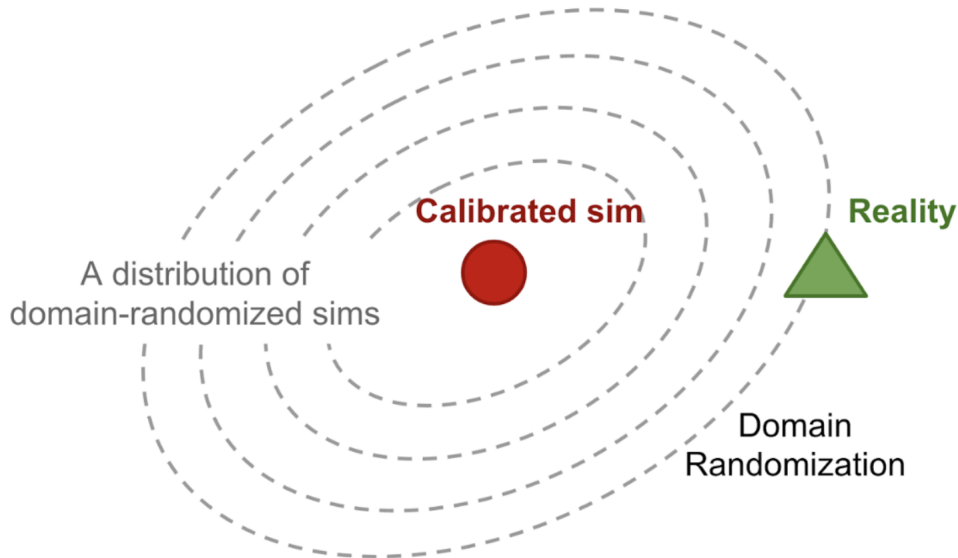


Figure 3.3: Illustration of the domain randomization concept. The physical domain is a possible sample from the distribution induced by parameters randomization [Weng 19].

tation when learning a task with a policy network. As a consequence, the unconstrained back-propagation procedure may result in an entangled knowledge representation. In this setting, it is difficult to determine which part of the network is relative to the task or to the agent, making a partial or total transfer of the policy network hazardous if done naively and with very low chances of success. We empirically demonstrate this point in Section 3.3.2. Moreover, control policies frequently leverage feed-forward neural networks where the input and/or output dimensions depend on the number of degrees of freedom (DoF) of an agent. As such, the first and the last layers cannot be directly transferred to an agent with a slightly different control dimensionality. Analogous to CV, it would be interesting to be able to pre-train a model on a given domain (i.e data-set for CV or agent for RL) such that it performs well or can be efficiently fine-tuned on another domain for the same task. As a practical example, we can think of a model pre-trained on Imagenet and then fine-tuned on a classification task for tumour detection [Saikat Islam 22], or in the context of RL, a model pre-trained on a 6 DoF UR10 robot and then transferred to a 7 DoF Panda robot for the same manipulation task.

3.3 Problem statement

In this section, we define and contextualize our problem by adapting the general definitions of Transfer Learning given in Section 3.2 to suit our specific setting.

3.3.1 Formalization

To formalize the CATL problem, we employ the formalization used in [Kim 20]. We define a *domain* as a MDP without a reward function R . Thus, it forms a 4 tuple written as $D = \langle S, A, P, \rho_0 \rangle$. In other words, a domain fully describes the agent embodiment

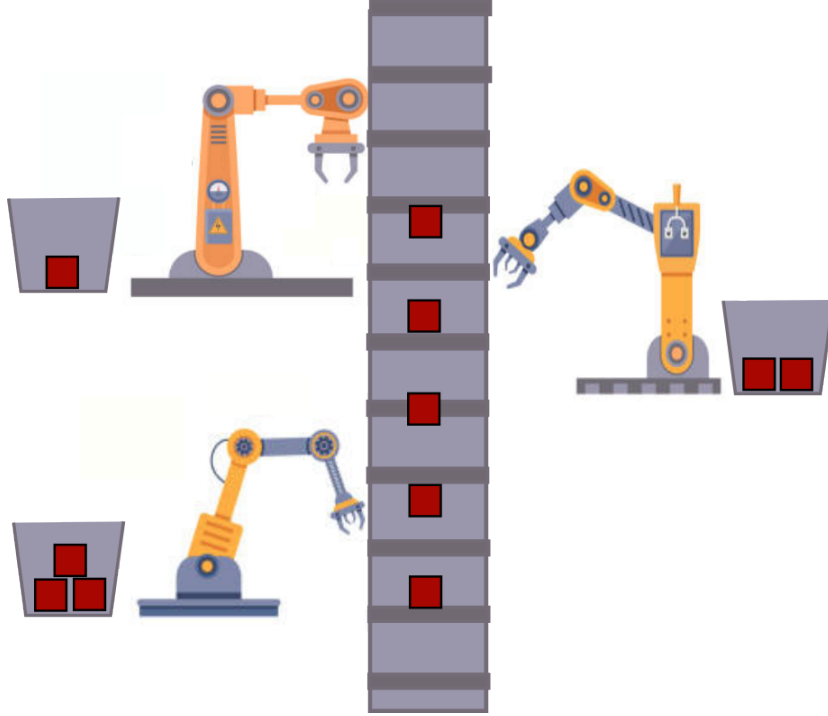


Figure 3.4: Illustration of the Cross-Agent Transfer Learning philosophy: 3 robots with different morphologies are performing the same task on an assembly line. The policy used to learn and run the pick and place task is robot-agnostic (i.e the same across robots).

and its dynamic, as well as the environment, but does not imply any particular behavior. The task reward $R_{r,\mathcal{T}}$ is the only component specifying how the robot should behave. Although our study is focused on robots, we believe that our problem formulation and the associated solution are sufficiently general to be applied to any kind of agents. As such, in this work we refer to domain, agent and robot equivalently. A task on the other hand, specifies the desired agent’s behavior and can be represented by the appropriate reward function R . Following the formalization proposed by [Devin 17], we define a world w as:

Definition 1: Given a domain $D_r = \langle S_r, A_r, P_r, \rho_{0,r} \rangle$ (i.e a robot r) and a task \mathcal{T} , a world $w_{r,\mathcal{T}}$, is a modular MDP combining a domain and a task such that:

$$w_{r,\mathcal{T}} = D_r \cup \mathcal{T} = \langle S_r, A_r, P_r, \rho_{0,r}, R_{r,\mathcal{T}} \rangle \quad (3.1)$$

Note that $R_{r,\mathcal{T}}$ depends on r because we may use robot-specific terms in the reward function to penalize large joint movements for instance.

Definition 2: A Universe U is the set of all possible worlds, or more formally, if we denote by \mathfrak{T} the set of all the tasks considered and \mathfrak{R} the set of all the domains considered:

$$U = \{w_{r_i,\mathcal{T}_j}\}_{i=1:|\mathfrak{R}|}^{j=1:|\mathfrak{T}|} \quad (3.2)$$

where $|\cdot|$ denotes the cardinality of a given set.

We define the cross-agent transfer problem that we are tackling as follows, slightly adapted from [Zhu 20]:

Definition 3: Given a set of n source worlds $\{w_{s_i, \mathcal{T}}\}_{i=1:n} \subset U$ and a target world $w_{t, \mathcal{T}} \in U$, Cross-Agent Transfer Learning (CATL) aims to derive an optimal policy $\pi_{t, \mathcal{T}}$ for the target world more efficiently than learning it from scratch, by leveraging information I_s^i from $\{w_{s_i, \mathcal{T}}\}_{i=1:n}$ as well as information I_t from $w_{t, \mathcal{T}}$.

The above definition is rather general but conveys the idea that we transfer knowledge from one or several source domains to a target domain with respect to a given task (see Figure 3.4). It also makes no assumption about the nature of the information I transferred. As such it could be a teacher neural network’s hidden state as in [Wan 20], Q-Values as in [Beck 22], expert demonstrations or a partial/complete policies. As discussed and analyzed in [Da Silva 19], it is not trivial to define what kind of knowledge should be transferred and through which medium. A recent but popular and successful type of approach to address CATL, is to find a single policy that can manage a variety of robot hardware configurations for a given task \mathcal{T} . In this case, what is transferred is the policy. More formally, if we denote by $\pi_{\mathcal{T}}^*$ such a policy and \mathfrak{R} the set of robot morphologies contained in our universe U , we wish to solve:

$$\pi_{\mathcal{T}}^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_{r \sim \mathfrak{R}} [G^{\mathcal{T}}(r)] \quad (3.3)$$

where $G^{\mathcal{T}}(r)$ is the discounted return for robot $r \in \mathfrak{R}$ on task \mathcal{T} . In other words, $\pi_{\mathcal{T}}^*$ should be optimal and robot-agnostic.

In this work, we will consider the challenging CATL setting where \mathfrak{R} is composed of robots functionally and morphologically different. In other words, they differ both by their segment length and by their number of joints. If we denote by D_s , the source domain and D_t , the target domain, then we will have in general $\dim(A_s) \neq \dim(A_t)$ and $\dim(S_s) \neq \dim(S_t)$.

3.3.2 A naive attempt

Given the current transfer learning paradigm in other ML fields such as NLP and CV, one might be tempted to use the same straightforward approach with TL in Reinforcement Learning. Given an expert policy network $\pi_{R_1, A}$ trained on task A with robot R_1 , we could attempt a direct policy transfer by re-using some of the layers of $\pi_{R_1, A}$ to initialize $\pi_{R_2, A}$, the student policy network for the same task using robot R_2 . Similarly to CV and NLP, only the newly added layers would be adjusted by the gradient back-propagation procedure. Intuitively, we can expect that some useful knowledge about how to achieve task A resides inside these transferred layers, which could jump-start and speed up training for $\pi_{R_2, A}$. We explore this simple idea in a toy experiment to demonstrate its inefficiency and highlight the need for a more sophisticated approach.

Experiments

We consider two serial robot arms: a UR10 robot with 6 DoF and a Panda robot with 7 DoF, both trained from scratch with PPO on a ball catching task. In this task, the agent is equipped with a basket and must catch a ball thrown at him. We denote by π_{E_P} and π_{E_U} the expert policy obtained respectively for the Panda and UR10 robots. To study the effectiveness of a naive policy transfer, we initialize new policy networks for both robots with the expert policy trained of the other robot, and train the new models. More

formally, we denote by $\pi_{S_P \rightarrow U}$ the UR10 student policy initialize with the expert policy trained on the Panda robot and $\pi_{S_U \rightarrow P}$ the reciprocal. Note that given the mismatch in control dimensionality due to different numbers of DoF, only the hidden layers of the policy networks can be transferred (see Figure 3.5). Given this setup, we analyze two cases: (A) only the randomly initialized layers (i.e input and output layers) are adjusted by the task gradient and (B) the whole policy network is fine-tuned.

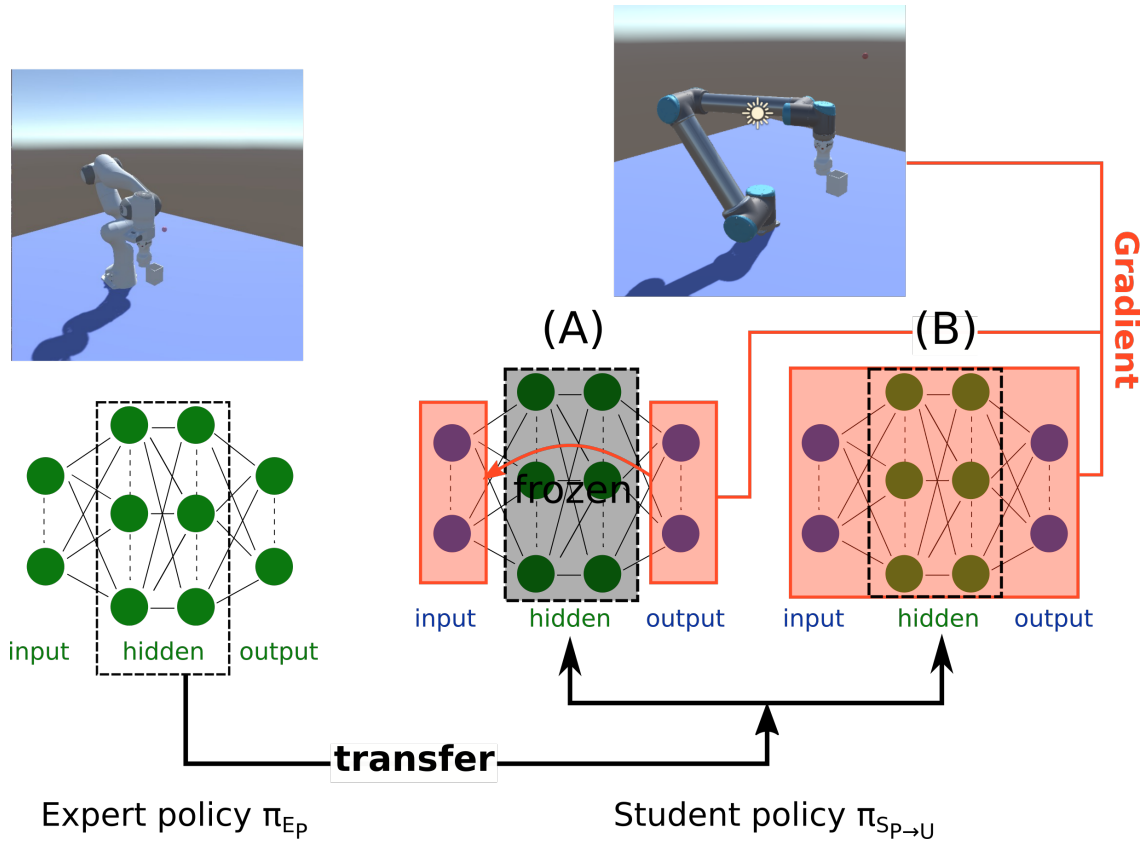
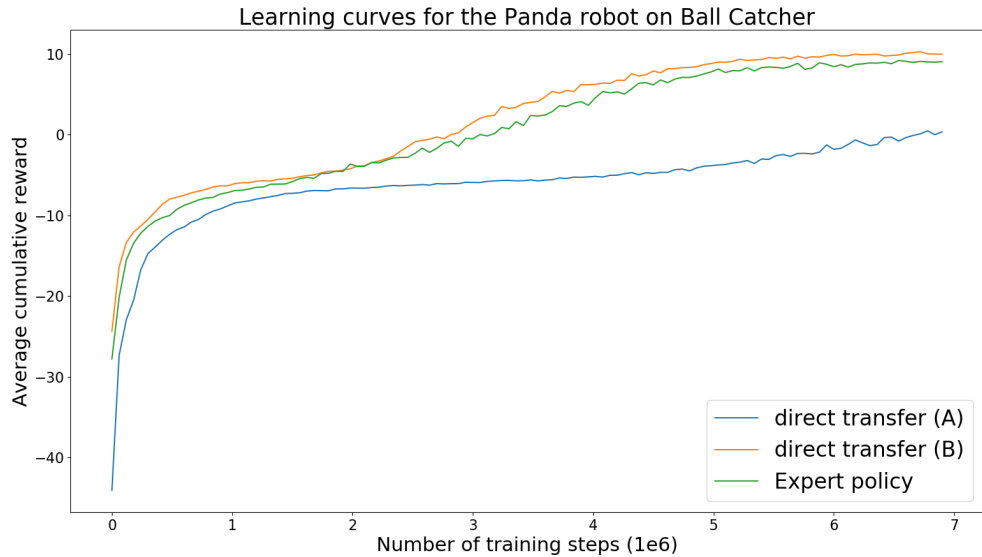


Figure 3.5: Illustration of the naive experiment setup. The hidden layers of the expert policy trained on the panda robot are used to initialize the student policy to be trained on the UR10. Only the input and output layers are initialized from scratch. When fine-tuning for (A) only the randomly initialized layers and for (B) the whole policy network.

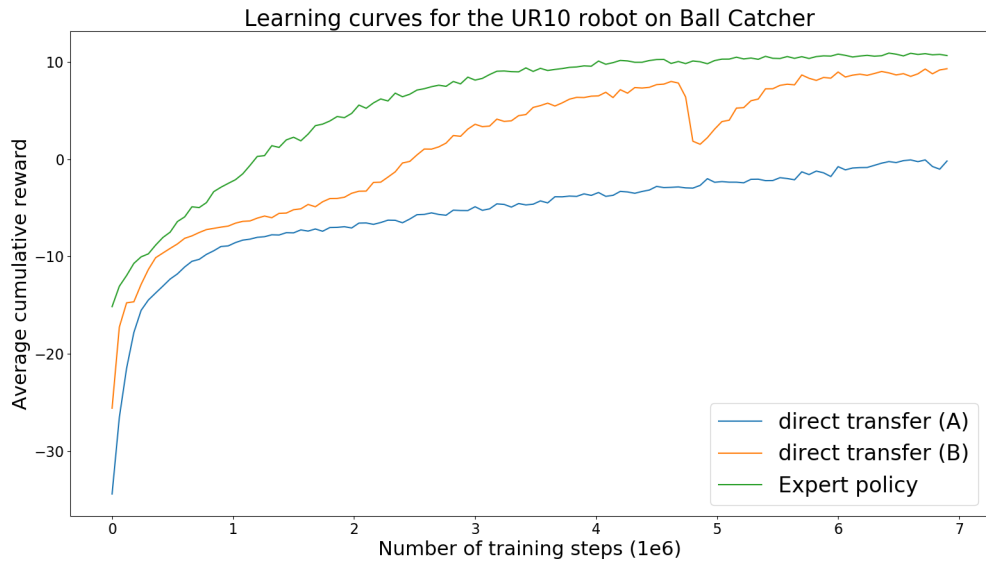
Results

We examine the results for both transfer settings: Panda to UR10 and UR10 to Panda. The learning curves are depicted in Figure 3.6. We can see the evolution of the mean cumulative reward obtained by the agent over time as training progresses. Therefore, it is a reliable metric to monitor the agent’s performance in its environment and a good indicator of learning efficiency. If direct transfer of the expert policy hidden layers was beneficial, we should observe the student agent converge to a high reward faster than the expert trained from scratch on the same robot. However, Figure 3.6 show that in both cases A and B, there is no improvement over regular training from scratch with PPO. In fact, case A is significantly slower to converge than baseline. These results indicate that a naive policy transfer not only fails to improve sample-efficiency, but may also prove to be detrimental for learning. This is likely due to a lack of structured knowledge in

the policy network, leading to an entangled task-specific/robot-specific representation. In contrast, we demonstrate the effectiveness of our approach on the same challenging setting in chapter 4.



(a) Panda robot.



(b) UR10 robot.

Figure 3.6: Learning curves when using direct transfer of the hidden layers between the panda and the UR10 robot with fine-tuning for (A) only the randomly initialized layers and for (B) the whole policy network.

3.4 Foundational Works

In this section, we provide a comprehensive review of the foundational works that inspired the development of our own related contribution, naturally leading the discussion towards a detailed description of all the components of our approach and design decisions in the next chapter. In particular, we highlight some of the limitations of these previous

Cross-Agent Transfer Learning techniques, thus justifying the need to develop our own approach to address these. We begin with an overview of two early attempts at CATL, namely *Transfer with Invariant Feature Spaces* and *Modular Network Policy*. Lastly, we focus our attention on the Universal Notice Network (UNN) framework as it is the main building block of our contribution. Furthermore, many of the concept explained for the UNN will also be relevant when we discuss the Latent-Space UNN in Chapter 4 and another contribution, named the Delay-Aware Universal in Chapter 5.

3.4.1 Invariant Feature Space

This approach was jointly introduced in [Gupta 17] by Colin Devin and Abhishek Gupta. The aim of their mutual work is to enable two agents, regardless of their potential morphology divergence, to learn multiple skills by sharing information. Their endeavor was also motivated by a desire to improve sample efficiency of RL algorithms and accelerate the acquisition of new skills.

Learning a shared latent space

Given two agents r_S and r_T , their methodology leverages the common skills learned by both agents to train an invariant feature space, which can then be used to transfer additional skills between them. Formally, let $\eta^{\pi_S^*}(s_S)$ and $\eta^{\pi_T^*}(s_T)$ denote, respectively, the state distribution induced by the optimal policies π_S^* and π_T^* of r_S and r_T . A common feature space $K = \{f, g\}$ is defined using two embeddings f and g such that $p(f(\eta^{\pi_S^*}(s_S))) = p(g(\eta^{\pi_T^*}(s_T)))$. In plain English, the embedded state distributions of the optimal policies are similar in the invariant feature space. To learn this shared latent space, they assume that the agents possess prior knowledge of one another in the form of basic, common skills, learned while performing a "proxy" task. Using this setup, they can compare how they perform this task in order to estimate the correspondences between both agents and determine the common feature space. In practice, they first obtain pairs of corresponding states across domains through time-alignment. In other words, they consider that both agents perform the same proxy task at the same pace, and therefore, states visited at the same time step can be matched. Additionally, the state space of each domain is divided into an agent-specific state x_r and a task-specific state o_r . Given a list of paired states, the embeddings f and g can be learned using a simple similarity metric introduced in [Chopra 05]:

$$L_{sim}(x_S^p, x_T^p) = \|f(x_S^p; \theta_f) - g(x_T^p; \theta_g)\|^2 \quad (3.4)$$

where subscript p denotes the proxy skill and θ_f and θ_g are the parameters of the neural network approximating f and g . Equation (3.4) can be trivially optimized by setting every embeddings to 0, i.e $f(x_{S_{p,r}}) = g(x_{T_{p,r}}) = 0$. As such, it is necessary to add reconstruction losses in the training objective, to ensure that useful information is encoded in the latent space.

$$L_{rec_S}(x_S^p) = \|x_S^p - Dec_S(f(x_S^p); \theta_{Dec_S})\|^2 \quad (3.5)$$

$$L_{rec_T}(x_T^p) = \|x_T^p - Dec_T(g(x_T^p); \theta_{Dec_T})\|^2 \quad (3.6)$$

In practice, this yields an auto-encoder training scheme where Dec_S and Dec_T respectively decode the states encoded by f and g . The full training objective is thus:

$$\min_{\theta_f, \theta_g, \theta_{Dec_S}, \theta_{Dec_T}} \sum_{(x_S^p, x_T^p)} L_{sim} + L_{rec_S} + L_{rec_T} \quad (3.7)$$

A diagram of their learning approach is shown in Figure 3.7.

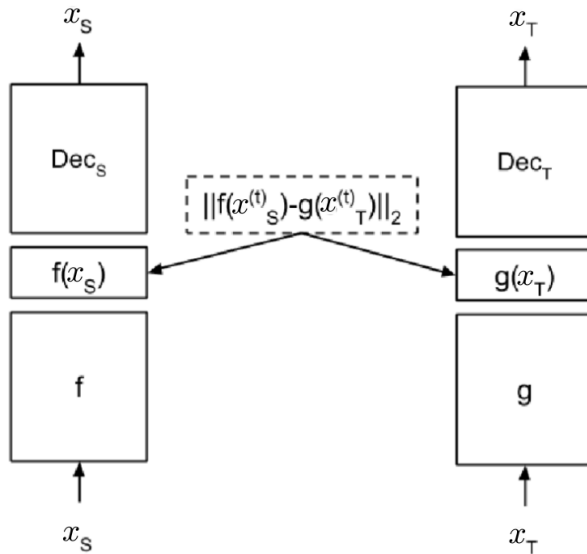


Figure 3.7: Auto-encoders training with an Euclidean distance between embeddings as the similarity metric[Gupta 17].

Transferring knowledge

Using the invariant feature space defined by f and g , they can transfer knowledge and speed up the learning process in the target domain. To do so, they incentivize the distribution of trajectories in the target domain to be similar to the source domain under the mappings f and g . This is done by adding a term $r_{transfer}$ to the reward function

$$r_{transfer} = \alpha ||f(x_S^{(t)}; \theta_f) - g(x_T^{(t)}; \theta_g)||^2 \quad (3.8)$$

where $x_S^{(t)}$ is the source agent-specific state at time step t under the optimal policy and $x_T^{(t)}$ is the target agent-specific state at the same time-step during training. The constant α is weighting the contribution of the transfer reward relative to the overall task goal.

$$r_t = r_{transfer} + r_{task} \quad (3.9)$$

where r_{task} is the task-specific reward. This shaped reward function provides an additional training signal that guides the learning policy in the target domain towards potentially beneficial states.

They assessed the effectiveness of their approach on simple planar robots (depicted in Figure 3.8) on several 2D manipulation tasks such as button push and block push. Overall, their method demonstrated an improved sample efficiency when compared to baseline agents trained without knowledge transfer. Moreover, it succeeded in learning tasks where r_{task} is sparse using the extra guidance offered by $r_{transfer}$. However, their method does not allow for zero-shot transfer and has not been shown to scale to more complex tasks and higher-dimensional robots.

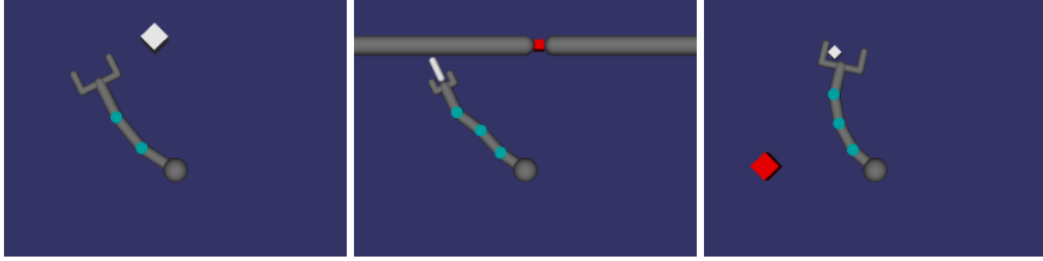


Figure 3.8: Robots considered for the experiments performing the proxy tasks [Gupta 17].

3.4.2 Modular Network Policies

This approach was also introduced by Gupta and Devin in 2017 [Devin 17]. The goal is similar: transfer knowledge between robots with different kinematic and dynamic structure to improve sample-efficiency. They also introduced the notion of *world* and *universe* used and adapted in Section 3.3.

Principle

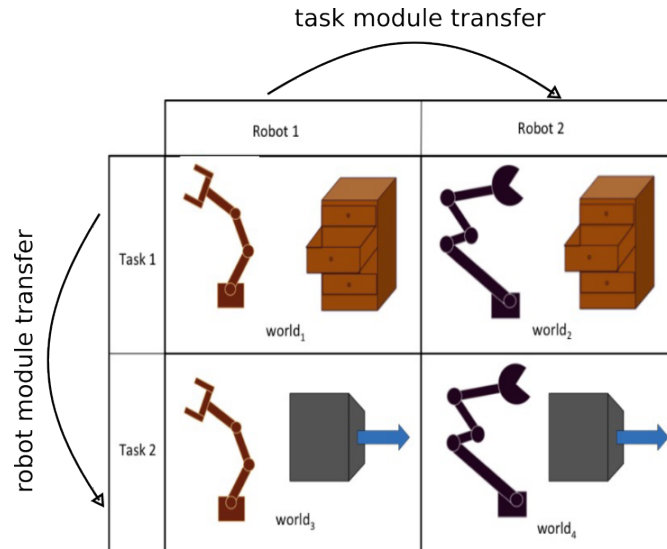


Figure 3.9: Illustrative example of a Universe with 4 different worlds (2 robots and 2 tasks) [Devin 17].

Their approach aims to demonstrate that neural network policies can be divided into two modules: "task-specific" and "robot-specific". The task-specific modules are shared across different robots, while the robot-specific modules are shared across all tasks performed by that robot (see Figure 3.9 for an illustration). This decomposition can be leveraged to share task-related information, such as perception, across different robots, and share robot-related information, such as dynamics and kinematics, across different tasks. As such, they denote by s_w the observation for world w . They consider that an observation can be split into "intrinsic" (i.e robot-specific) information $x_{w,r}$ and "extrinsic" (i.e task-specific) information $o_{w,\tau}$. Robot-specific information $x_{w,r}$ could include joints state, images and sensor readings, while task-specific information $o_{w,\tau}$ may contain object

locations and position of the robot’s end-effector for instance. Furthermore, let $\pi_{w_r, \mathcal{T}}(\cdot|w)$ be the policy for world w with robot r performing task \mathcal{T} such that:

$$\pi_{w_r, \mathcal{T}}(\cdot|w) = \mathcal{N}(\phi_{w_r, \mathcal{T}}(o_w), \Sigma) \quad (3.10)$$

with $\phi_{w_r, \mathcal{T}}$ a neural network and Σ a diagonal covariance matrix. The function $\phi_{w_r, \mathcal{T}}$ is the composition of the robot-module f_r and the task module $g^{\mathcal{T}}$. Given above definitions, we can formally express the modular policy with:

$$\phi_{w_r, \mathcal{T}}(o_w) = \phi_{w_r, \mathcal{T}}(o_{w, \mathcal{T}}, x_{w, r}) = f_r(g^{\mathcal{T}}(o_{w, \mathcal{T}}, x_{w, r})) \quad (3.11)$$

To enable the reuse of modules across similar instances, a distinct set of parameters is used for each robot and task module. Specifically, if multiple worlds feature the same robot instantiation r , they would utilize the same robot module f_r , while worlds that share a task instantiation \mathcal{T} would employ the same task module $g^{\mathcal{T}}$.

Learning Modular Policies

Given a robot module and a task module, the expectation is that their combination will produce a fully operational policy. This implies that both modules are able to generalize to each other, even though they have not been trained together. To reach this goal, the authors proposes to train simultaneously mix-and-match modules such that a common feature space eventually emerges through ongoing interactions between a wide variety of modules. As the number and diversity of worlds grow, the likelihood of a robust and general shared feature space emerging also increases. Consequently, the trained modules become increasingly invariant to variations in tasks/robots, which also improves generalization to novel worlds. The training procedure is thus conceptually simple: train multiple robot and task modules simultaneously until a global optimum is reached (with respect to the tasks reward functions). The training procedure is illustrated in Figure 3.10.

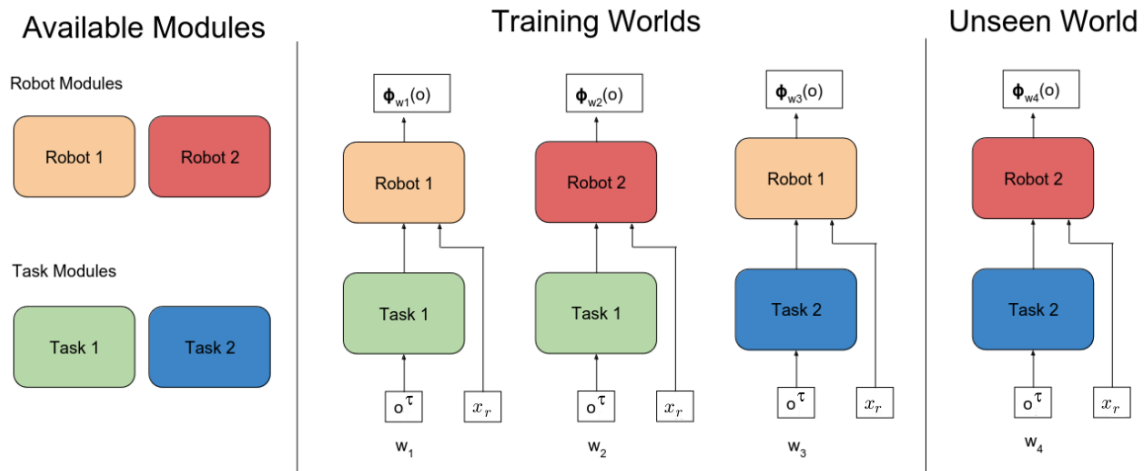


Figure 3.10: This universe is composed of 2 robots and 2 tasks with an equal number of modules. A subset of these modules is drawn and combined to be trained jointly until convergence. One of the possible combinations has been left off to serve as a test world [Devin 17].

They thoroughly tested their approach with a multitude of 2D robots and manipulation tasks (see Figure 3.11). They report an improved sample efficiency as well as zero-shot generalization in some cases. However, it is not clear how much time is needed to jointly train every combination. This could potentially be prohibitively large for more complex robots and tasks, significantly exceeding the time needed for a regular training on the desired world. Moreover, it is unknown how many training worlds are required to reach a shared and robust latent representation for efficient transfer.


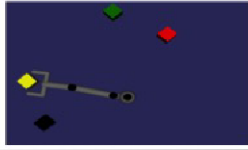




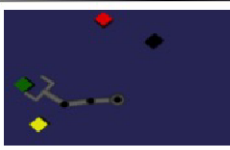


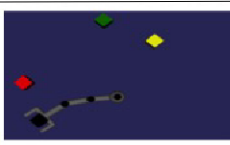

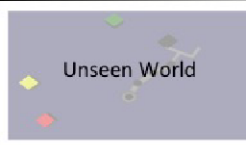
Robots Tasks	3link	3link different config	4link
Reach Yellow			
Reach Red			
Reach Green			
Reach Black			

Figure 3.11: Grid of tasks and robots considered for their experiments. As shown, one world was not seen during training and later tested on to study generalization [Devin 17].

3.4.3 Universal Notice Network

Our primary source of inspiration and one of the earliest attempts of CATL is the Universal Notice Network method (UNN) [Mounsif 19b], defined and explored by Mehdi Mounsif during his PhD [Mounsif 20].

Principle

The main motivation behind the UNN framework was to tackle the entangled knowledge representation issue resulting from the unconstrained optimization procedure in the shallow networks typically used in Reinforcement Learning. As such, it aims at decomposing and segmenting the knowledge encompassed in a policy network such that it can be efficiently transferred to multiple agents, regardless of their morphologies, number of articulations or actuators. This approach is based on a functional and hierarchical decomposition of the policy, with at the center, a high-level task-specific module, and on each side, acting as an interface, agent-specific modules (see Figure 3.12). The task-module is called the Universal Notice Network to highlight the idea that it should contain a set of

high-level instructions that any kind of robot could follow to solve a given task. This is analogous to a notice used to build furnitures where the instructions are general enough to be achievable by most people as they do not assume any particular morphologies. In other words, the UNN module is not concerned with the details of actuation and focuses on the high-level decisions. As such, the task-solving process is largely agnostic to the robot morphology and can suit any of the targeted robots for transfer. For example, solving a pick and place task may require low-level robot-specific motor commands, but the high level process will roughly be the same: first move the effector close to the pick target, grasp the object and get it at the desired location. Consequently, the agent-agnostic nature of the UNN module makes it suitable for transfer between agents, even in the presence of morphology discrepancies.

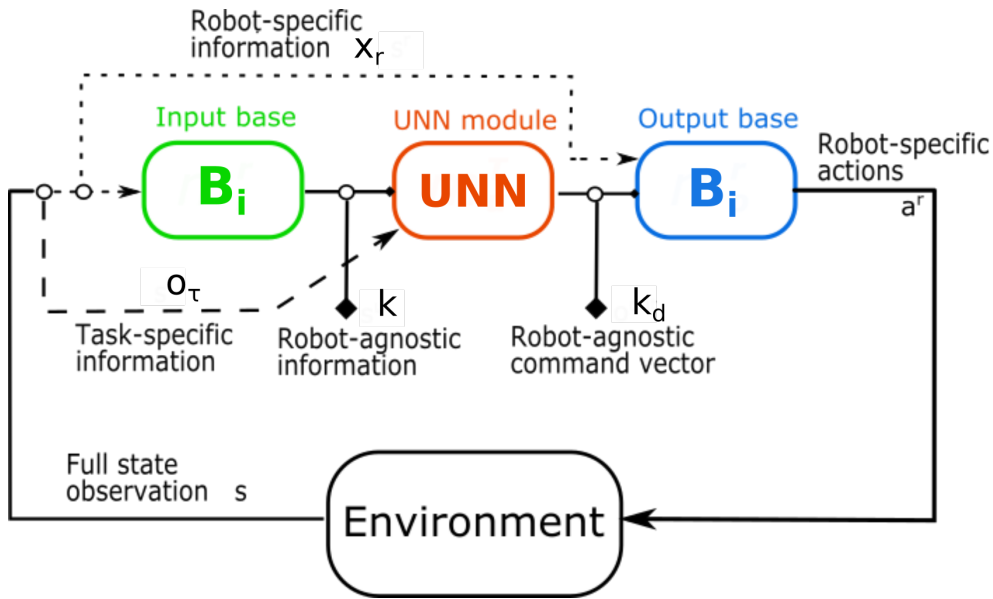


Figure 3.12: Schematic representation of the UNN pipeline[Mounsif 23] and its three different modules. The red task-specific module is at the center with the robot-modules on each side

However, it is primary to ensure that the agent has the mobility and capacities required to comply with the UNN instructions and accomplish the task. Therefore, the UNN methodology assumes indeed that the agent can perform the required actions. Formally, if R denotes the set of feasible actions the robot can produce and U the set of actions required by the UNN to perform a task, then it should be satisfied that:

$$U \subseteq R \quad (3.12)$$

Another prerequisite to enable transfer is the availability of robot-specific modules called the bases in the UNN framework. They are used as intermediaries between the high-level UNN task-module instructions and the agent's low level hardware. More specifically they serve two main purposes:

- Translating robot-specific observations into a feature space shared by the considered agents before being fed to the UNN module. This part is handled by the input base (green module in the Figure 3.12).

- Translating the high-level and robot-agnostic commands delivered by the UNN module into robot-specific actions that the robot can execute (i.e robot’s actuation). This mapping is managed by the output base (blue module in Figure 3.12).

Similarly, the bases modules are designed to be task-agnostic and can be reused with other tasks. However, they are unique to a robot hardware and morphology. A complete and fully functional policy thus requires to combine the corresponding robot modules and task module as presented in Section 4.3. When varying the task, the robot modules are kept unchanged and when varying the robot we transfer the task module. Figure 3.12 illustrates the full policy architecture with the corresponding input/output of each module.

Formalization

More rigorously, the three modules form a pipeline composed of the input base B_r^i and in between the output base B_r^o , specific to the robot, and the UNN $U_{\mathcal{T}}$ agent-agnostic and specific to the task only. In this modular framework, the full observed state s provided by the environment is split into robot-specific and task-specific information:

$$s = \{x_r, o_{\tau}\}$$

The robot-specific information x_r holds the observations regarding the state of the robot itself, such as joint position, or joint velocity. In contrast, everything the agent needs to know about the state of the task is contained in the task-specific information o_{τ} . It could include objects location and velocity and more broadly any robot-agnostic data. We denote by K the shared feature space in which the UNN module operates. Please note that the UNN formulation is rather general and does not suppose any particular nature for the feature space. The input base $B_r^i : X_r \mapsto K$ maps the robot state to the feature space K while the output base $B_r^o : K_d \mapsto A_r$ remaps the UNN command to the robot action space.

To summarize, at each time step, the environment produces s_r , the full state which is first decomposed into robot specific information x_r and task specific information o_{τ} . Then, the robot specific vector x_r is translated into a robot agnostic vector $k \in K$ by means of the input base

$$k = B_r^i(x_r) \tag{3.13}$$

Following this, the UNN processes the agent-agnostic representation k as well as o_{τ} , task-related information, to compute a robot agnostic action k_d , which can be seen as the desired robot action in the feature space:

$$k_d = U_{\mathcal{T}}(k, o_{\tau}) \tag{3.14}$$

Finally, we get the effective action executed by the robot a_r in the environment by projecting the action from the common feature space $k_d \in K_d$ back to the robot space through the use of the output base

$$a_r = B_r^o(k_d) \tag{3.15}$$

The modular policy $\pi_{r,\tau}^*$ is thus the composition of the 3 modules

$$\pi_{r,\tau}^* = B_r^o \circ U_{\mathcal{T}} \circ B_r^i \tag{3.16}$$

or more explicitly

$$\pi_{r,\tau}^* = B_r^o(U_{\mathcal{T}}(B_r^i(x_r), o_r)) \quad (3.17)$$

This modular and functional decomposition approach also allows us to overcome a technical difficulty: since we are dealing with transfer for robots with heterogeneous state and action dimensions (i.e $\dim(A_s) \neq \dim(A_t)$ and $\dim(S_s) \neq \dim(S_t)$), we cannot have a single policy fitting every state-action space dimension. This issue is solved by having each robot represented by its own pair of bases, fitted to its control dimensionality.

Modules training

In practice, each of the three mappings discussed above B_r^i , B_r^o and $U_{\mathcal{T}}$ can be either learned by a feed-forward neural network or obtained via analytical methods. In the "classical" UNN framework ([Mounsif 19a], [Mounsif 23]), the robot-agnostic information used by the UNN module has to be "manually" defined. For instance, in his experiments, Mehdi Mounsif sets the feature space to be the Cartesian space (i.e $K = \mathbb{R}^3$) of the end-effector. As such, the UNN module is working either with the end-effector position or velocity.

Before training or transferring a UNN module, it is first mandatory that the source or target agent has the infrastructure to comply with the UNN instructions. Therefore a pair of bases corresponding to the considered robot should be either already available or trained prior to using/training a UNN module. In the case where bases are obtained using neural networks, they can be trained on a suitable primitive task to acquire basic motor skills. Another alternative is to collect a data-set of trajectories of the robot and fit a regression model with supervised learning techniques. Finally, the last option consists in using analytical models for the robot bases. Regardless of the procedure used to obtain the robot modules, the input and output bases will respectively serve as forward and inverse kinematic models of the agent, since we are dealing with the Cartesian space, i.e a_r is a desired position.

Naturally in this paradigm, the UNN module (or task module) has to output the desired effector position (or velocity) given task-related information and current effector position (or velocity). It can be trained via Reinforcement Learning in two different ways. In the first case, the UNN is coupled with a robot and its associated bases. The UNN interacts with the environment through the bases and its error on the task is back propagated through the network. However, the UNN module may then take advantage of the robot hardware structure to achieve the task (for instance, blocking an object between two articulations). As a consequence, the UNN may favor certain body configurations which may be detrimental for transfer. This issue is tackled by the Base Abstracted Modeling (BAM) method [Mounsif 19a] also developed by Mehdi Mounsif. It assimilates the robot to its effector by setting B_r^i and B_r^o to identity mappings, thus making no assumption on the robot's constitution and preventing any bias related to the bases (see Figure 3.13). This is equivalent to considering a purely virtual and free-flying robot. BAM has shown, in practice, a faster convergence of the policy and a more defined knowledge segmentation, which in turn improves UNN transfer. In both cases, the stochastic gradient ascent procedure focuses on the UNN module's weights, leaving the bases unaffected by gradients. An illustration of the difference between the "classical" UNN and BAM formulation is provided in Figure 3.13.

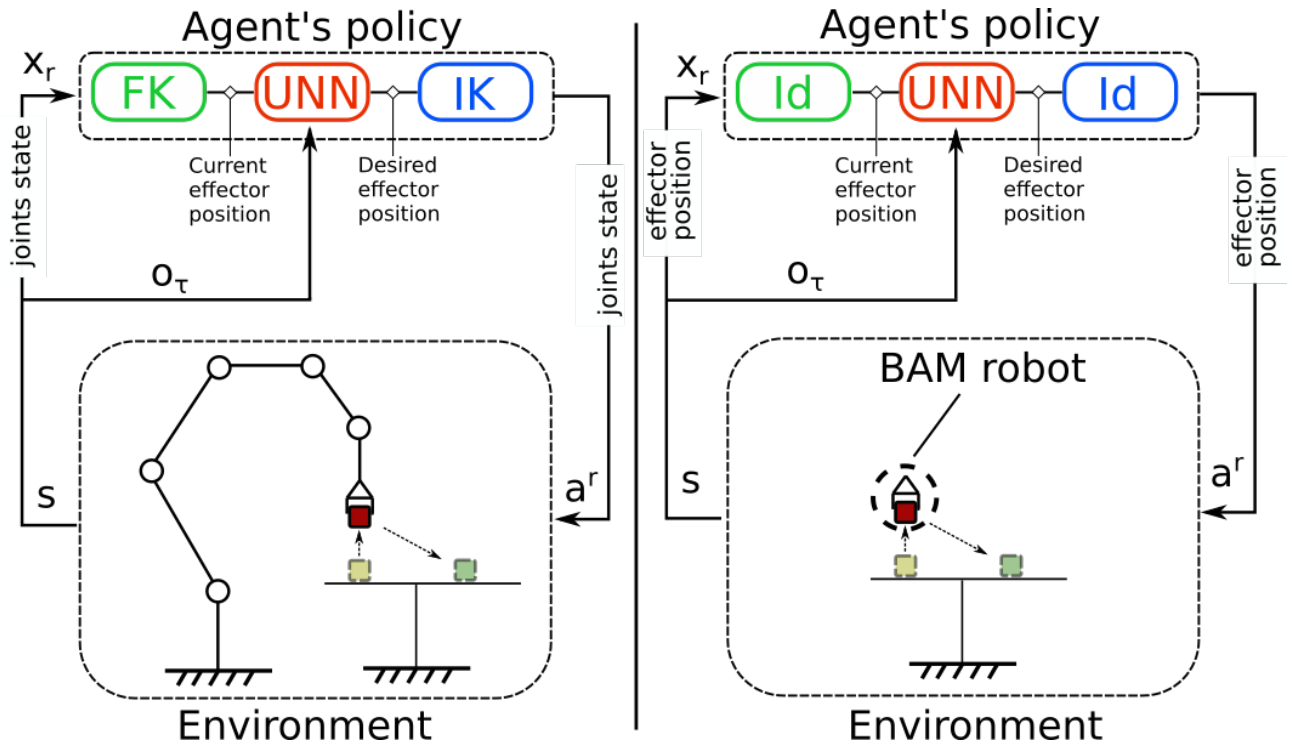


Figure 3.13: Schematic representation of the UNN module training with and without the Base Abstracted Modeling on a pick and place task. On the left, the "classical" UNN interacts with the environment through the robot bases represented by its kinematic models. On the right, the robot is assimilated to its effector and can move freely in the space, avoiding any bias due to the robot morphology.

Transfer is then as simple as a plug-and-play of the UNN module in between the robot's bases. In a number of cases, UNN transfer results in an instant generalization to a new and unseen morphology. In the worst case scenarios, performance achieved on the source robot can be recovered very fast with a bit of fine-tuning on the target robot. It is then possible to build a library of UNN modules and robot's modules, draw any subset of interest from it and combine a UNN/Bases pair into a novel fully functional policy.

This approach was exhaustively validated in simulation on a broad range of challenging manipulation tasks and robots. In every instance, the UNN-based agents significantly outperformed the baselines and demonstrated near zero-shot generalization. However, the efficiency of this approach was not proven on physical hardware. Figures 3.14 and 3.15 depict some of the robots and a task on which the UNN method was tested.

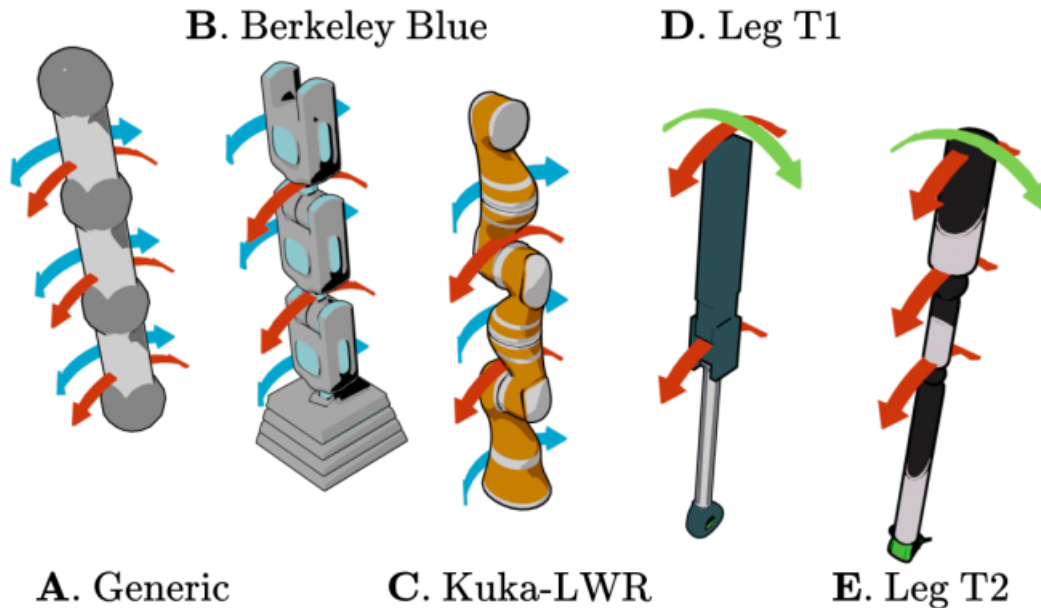


Figure 3.14: Schematic representation of some the robots used for the experiments on the UNN method. From left to right: Generic-3 robot, Berkeley Blue, Kuka-LWR, Leg Type 1, Leg Type 2. Image taken from [Mounsif 23]

3.5 Conclusion

This chapter introduced and formalized the concept of Transfer Learning. It discussed some of its most widely spread applications, in particular computer vision and natural language processing. Following this, this chapter formally defined and framed our specific problem setup: cross-agent transfer learning where the goal is to transfer knowledge between different agents in a RL setting. We first tried to solve our CATL problem with a naive, but straightforward method inspired by current approaches in CV and NLP. However, this simple approach proved to be inefficient at best, motivating the need for more sophisticated methods. Some of these approaches, discussed and analyzed in this chapter, yield very interesting results from a sample-efficiency standpoint, but either do not allow for zero-shot transfer or require a relatively high computing power. Lastly, we reviewed the original Universal Notice Network, a general framework for CATL, and detailed one particular instantiation where the agent-agnostic feature space is defined as the Cartesian space. However, in its current form, the UNN relies on hand-crafted features to define the space that interfaces the task module with the robots. As a result, this limits the method flexibility and practicability. To address this shortcoming, a new UNN approach was developed to learn the robot-agnostic feature space from multiple agents with different morphologies, thus removing the need for hand-crafted vectors. In the next chapter, this novel method is explained in details along with the corresponding experiments.

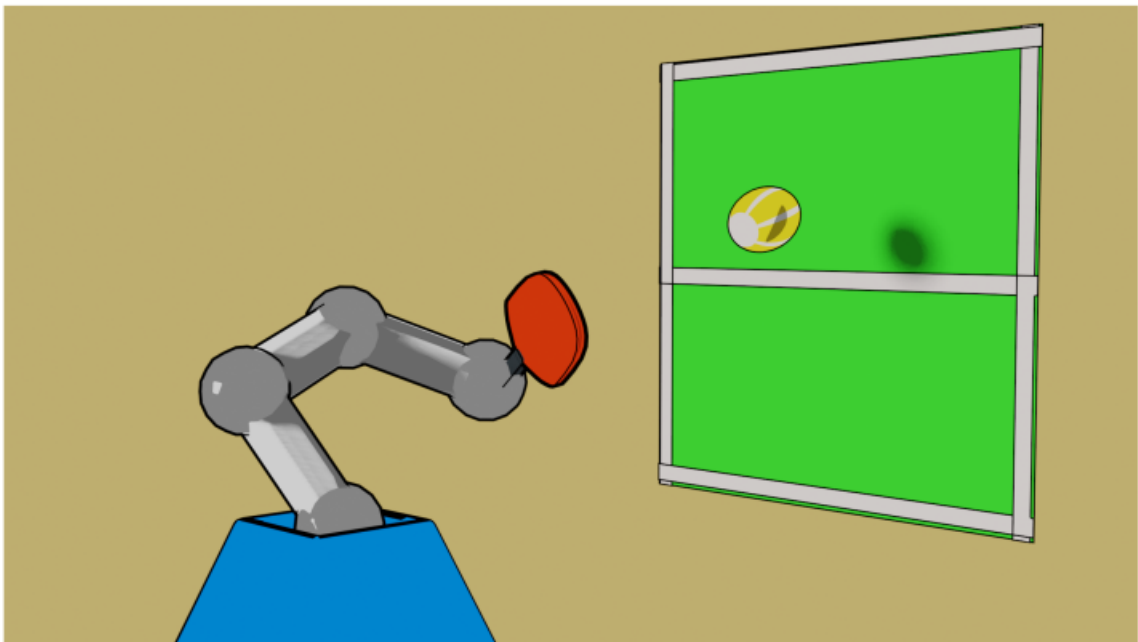


Figure 3.15: Schematic representation of a tennis-table setting, one of the tasks used for the experiments on the UNN method. Image taken from [Mounsif 23]

Chapter 4

Latent Space Universal Notice Network

This chapter introduces the Latent Space Universal Notice Network (LS-UNN), our main contribution. The motivation for our proposed framework stems from the limitations of several existing approaches, which we recalled and discussed in the previous chapter. To address these limitations, we have developed a methodology that builds upon the strengths of previous works and incorporates new strategies to overcome the shortcomings. In the dedicated results section, we present extensive experimental results supporting its effectiveness. Finally, we analyze how the LS-UNN fits into the current literature on CATL and draw connections to other related fields such as domain adaptation, imitation-learning and hierarchical reinforcement learning.

4.1 Motivations

One question left unanswered by the UNN framework in its original formulation, is how do we choose K , the shared feature space inhabited by the task-module? While the Cartesian space seems to be well suited for tasks involving robot arms, its definition relies on expert knowledge. Furthermore, it is not obvious what kind of feature space should be picked for more sophisticated and complex robots such as mobile base, robotic hands or bipedal robots. Even if an expert can manually craft a feature space, it is not guaranteed that this is the optimal agent-agnostic representation to support transfer of knowledge. Ideally, we should have a way of automatically learning an adequate representation for our agent-agnostic feature space, directly from raw data. This would avoid the need to rely on domain-specific knowledge or to assume particular properties of the robot morphology such as whether or not it possesses an end-effector. Moreover, given that we generally do not have access to the ground truth feature space, learning should proceed without supervision. Luckily, it is possible to leverage representation learning to discover the appropriate set of features for an efficient knowledge transfer. Furthermore, as explained in Section 2.6, feature learning frequently outperforms feature engineering in a number of tasks. This realization motivated our search for a unsupervised representation learning method that would generalize the "classical" UNN approach with a learned rather than manually defined, shared feature space.

4.2 Towards Zero-Shot Cross-Agent Transfer Learning via Aligned Latent-Space Task-Solving

In this section, we describe our CATL method, the Latent Space Universal Notice Network (LS-UNN), which extends the range of applications of the UNN framework. More specifically, it enhances the UNN formulation by providing a methodology for learning an agent-agnostic feature space, rather than relying on hand-crafted features. Our proposed method leverages Variational Auto-Encoders to learn an agent-agnostic latent space from paired, time-aligned trajectories collected on a set of agents of interest. We first provide a detailed and formal description of each component of the pipeline. Following this, we explain how the different modules come together to enable near zero-shot cross-agent transfer learning. Finally, we delve into the training process for each individual part and provide experimental evidence for the effectiveness of our CATL approach.

4.2.1 Preliminaries

As mentioned in the previous sections, we seek to learn the agent-agnostic feature space needed for cross-agent UNN transfer. Naturally, an ideal candidate for our goal is the auto-encoder framework described in Section 2.6.2. Indeed, its properties perfectly match the expected requirements:

- The encoder maps from the data space into a latent representation Z . Analogously, an input base maps from the robot space to a robot-agnostic feature space K .
- The decoder remaps from the latent space Z back to the data space. Similarly, an output base translates a robot-agnostic feature K .

From these observations, it is obvious that we could leverage a learned auto-encoder latent space for the robot-agnostic feature space given their similarities. As such, the UNN module, plugged between the bases, would operate within Z to learn the desirable latent state of the robot given a task (see Figure 4.1 for the full policy network). However, setting $K = Z$ raises a question. How do we ensure that the the latent representation Z is shared by the agents considered for transfer ? Simply learning independently a latent representation for each robot will very likely result in dissimilar latent spaces. In contrast we strive for a shared feature space, allowing the UNN module to learn a robot-agnostic policy. This requires to align the representations learned by the distinct robot’s bases. To address this issue, we rely on a time alignment procedure to find pairs of corresponding states across robots morphologies. We explain our approach and detail our entire transfer pipeline in the following subsections.

State pairing

Our solution to align the latent representation learned by all the considered robots is based on state-pairing (illustrated in Figure 4.2). Inspired by prior works [Gupta 17], we construct Z such that a pair of similar robot states from two different robots r_1 and r_2 (x_{r_1}, x_{r_2}) $\in X_{r_1} \times X_{r_2}$ maps to the same point in the latent space. As such, the LS-UNN workflow first requires to get corresponding pairs of states across robots morphologies. In this work, we considered *primitive* tasks performed by both robots at the same speed with optimal policies. The optimal policies can be obtained through standard analytic

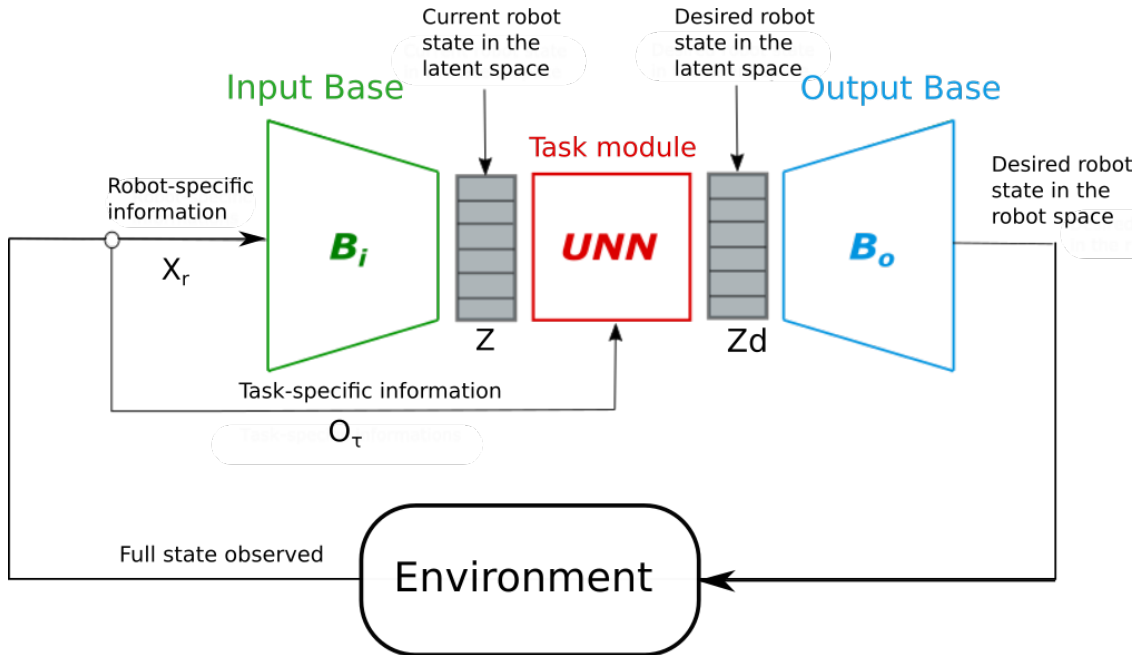


Figure 4.1: Schematic representation of the Latent Space UNN pipeline and its three different modules. The red task-specific module is at the center, inside the latent space, with the robot-modules on each side

methods or can be learned. Then, we assume that two states visited at the same time-step during the execution of the optimal policies for the primitive task can be matched. In other words, if we denote by $\pi_{r_1}^{*,\mathcal{T}}$ and $\pi_{r_2}^{*,\mathcal{T}}$ the optimal policy for primitive task \mathcal{T} on respectively, robot r_1 and robot r_2 , then:

$$x_{r_1}(t_1) \approx x_{r_2}(t_2) \iff t_1 = t_2 \quad (4.1)$$

with t_1 and t_2 two time-steps occurring during the same task execution and " \approx " denote similarity between two states. This simple but effective time-alignment process has already been used in prior works ([Gupta 17], [Makondo 18], [Makondo 15]), to find correspondences between states. In this work, we focus on efficient transfer of skills rather than how to find similarities between states. Finding correspondences using paired and time-aligned trajectories can be quite limiting and unpractical. Therefore, we study a more versatile approach over time-alignment in Appendix A.

In practice, we can consider several distinct primitive tasks (each with its optimal policy) to generate a large variety of trajectories along which states will be paired. This is to ensure that the robots bases generalize well even if they have not seen the entire robot state-space. As a consequence, these primitive tasks should be carefully crafted in order to be representative of how the robots can move, but also fairly simple to obtain optimal policies easily. It should also be taken into consideration that the bases training should focus on the state-space regions that will be visited during the UNN training for maximum efficacy. Once we have a dataset of corresponding paired states defined as:

$$D = (x_{r_1}^n(t), x_{r_2}^n(t))^{t=1:T, n=1:N} \quad (4.2)$$

where T is the length of a trajectory and N the number of trajectories, the bases can be trained in an unsupervised setting to build the shared latent space as described in the next section.

Primitive task execution

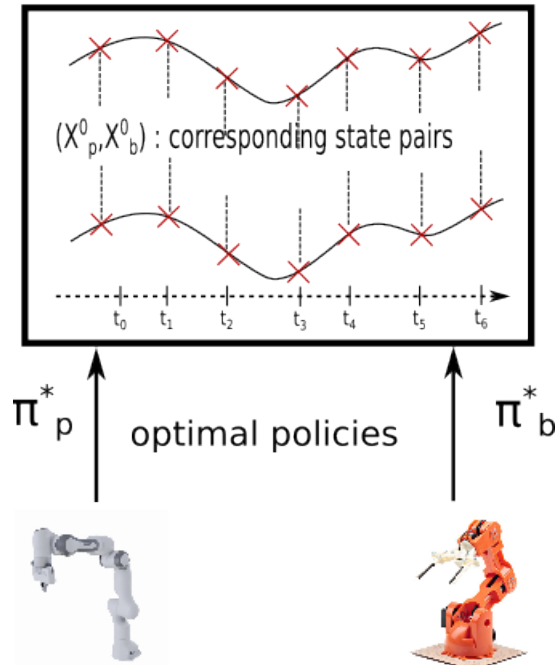


Figure 4.2: Illustration of the state pairing procedure for two robots with different morphologies.

4.2.2 Modules Training

The overall formalization and principle for the LS-UNN are very similar to the ones described in section 3.4.3. Indeed, equations and pipeline for the LS-UNN can be derived by simply replacing the robot-agnostic feature space K with the latent space Z and keeping everything else as is. The originality of our approach lies in the bases training procedure which encompasses the latent space building. In this subsection, we explicit the latent space creation and alignment process, as well as the task-module training process. First, we consider the case where we only have two robots at our disposal. But later in the same subsection, we describe the procedure to "plug" any extra robot to the common latent space already created to act as a source or target robot for UNN transfers.

Bases Training

In this Section, we explicit how to create and align the robots latent space Z . For this purpose, we assume access to D , a dataset containing pairs of corresponding states across the two robots considered, obtained following the procedure described in section 4.2.1. Figure 4.3 depicts the training procedure. From a high-level point of view, three essential conditions must be met at the end of training to increase the odds of zero-shot generalization for UNN transfers:

Condition 1: The latent space of both agents should be aligned. In other words, a pair of similar states $(x_{r_1}, x_{r_2}) \in D$ should map to the same latent vector z through their respective input bases.

Condition 2: Once decoded by the output bases of both agents, a latent vector z should give similar states $(\hat{x}_{r_1}, \hat{x}_{r_2}) \in D$ with $\hat{x}_{r_1} \approx x_{r_1}$ and $\hat{x}_{r_2} \approx x_{r_2}$.

Bases training

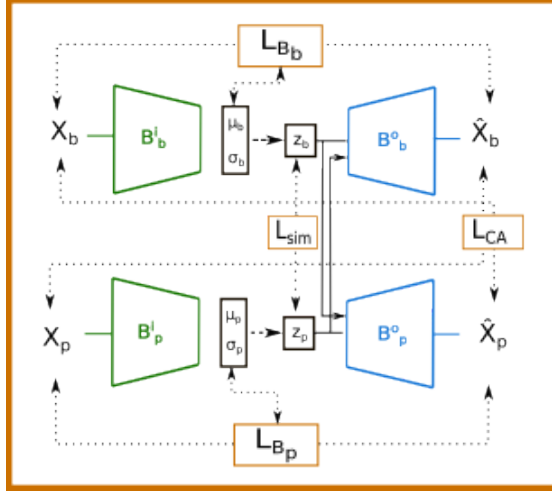


Figure 4.3: Illustration of the bases training procedure for two robots with different morphologies.

Condition 3: Finally, both metrics have to be balanced with a proper regularization of the latent space with the KL divergence to improve generalization.

Conditions 1 and 2 ensure that the latent spaces of considered agents are well aligned. This is paramount because the UNN module will output latent vectors and expect both agents to react similarly given the latent command. Condition 3 avoids "holes" in the latent space that would lead to meaningless decodings and as such, dissimilar states.

The two bases $B^i : X_r \rightarrow Z$ and $B^o : Z \rightarrow X_r$ are represented using variational autoencoders (VAE) structure to benefit from its dimension reduction and generative capabilities. They are specific to the robot so we will denote them by $B_{\phi_r}^i, B_{\theta_r}^o$, where subscript r denotes the robot and ϕ and θ respectively parametrize the encoder and decoder. In this regard, we rewrite Equation (2.44) with corresponding notations:

$$L_{B_r} = -\mathbb{E}_{z \sim B_{\phi_r}^i(\cdot|x_r)} [\log B_{\theta_r}^o(x_r|z)] + \beta D_{KL}(B_{\phi_r}^i(z|x_r) || \mathcal{N}(0, I)) \quad (4.3)$$

where x_r denotes the robot state (e.g joint velocity and joint position). As mentioned earlier, we wish to find a shared latent space between the robots by aligning their respective latent space to enable cross robot transfer. In other words, if $x_{r_1} \approx x_{r_2}$ (x_{r_1} similar to x_{r_2}), $z_{r_1} \sim B_{\phi_{r_1}}^i(\cdot|x_{r_1})$ and $z_{r_2} \sim B_{\phi_{r_2}}^i(\cdot|x_{r_2})$, we ideally want $z_{r_1} = z_{r_2}$ (similar inputs should be represented by the same latent vector). We enforce this condition through the use of a similarity loss defined as

$$L_{sim} = \mathbb{E}_{z_{r_1} \sim B_{\phi_{r_1}}^i, z_{r_2} \sim B_{\phi_{r_2}}^i} [||z_{r_1} - z_{r_2}||^2] \quad (4.4)$$

which encourages their encoding distance to be small. The reconstruction and the similarity losses ensure that both latent spaces are aligned and meaningful. Together they make sure that conditions 1 and 2 are satisfied. Condition 3 is enforced with the KL penalty. Each term should be carefully weighted as in practice, a low KL divergence is hard to conciliate with a low reconstruction error. Too much regularization will lead the latent space to collapse to the prior, hurting the reconstruction process and limiting the amount of information contained in the bottleneck. Conversely, not enough regularization will harm its generative capabilities.

Inspired by [Schönfeld 19], we also used a cross-alignment loss to further improve cross-domains transfers. Each decoder reconstructs its input by using the latent encoding of the paired similar state sampled from the other robot’s encoder:

$$L_{CA} = \mathbb{E}_{z_{r_1} \sim B_{\phi_{r_1}}^i} \left[\|B_{\theta_{r_2}}^o(z_{r_1}) - x_{r_2}\|^2 \right] + \mathbb{E}_{z_{r_2} \sim B_{\phi_{r_2}}^i} \left[\|B_{\theta_{r_1}}^o(z_{r_2}) - x_{r_1}\|^2 \right] \quad (4.5)$$

Finally, we train the bases of both robots end to end at the same time with the following full objective

$$\min_{\theta_{r_1}, \theta_{r_2}, \phi_{r_2}, \phi_{r_1}} \sum_{(x_{r_1}, x_{r_2}) \in D} L_{B_{r_1}} + L_{B_{r_2}} + \delta L_{sim} + \lambda L_{CA} \quad (4.6)$$

where δ and λ are constants respectively weighting the contributions of L_{sim} and L_{CA} to the full training loss.

Adding a new robot to the set

Even though we considered only 2 robots until now, it is possible to transfer a UNN module to and/or from an arbitrary number of robots that were not initially considered during the shared latent space building. There is no need to retrain any of the previously obtained robot modules. We simply need to align the new robot’s latent space to the already existing common latent space. The alignment process is depicted in Figure 4.4. The steps are very similar to the previously described workflow:

- Enlarge the dataset D of paired similar states with collected states on the new robot by following section 4.2.1.
- Use the input base $B_{\phi_{ref}}^i$ of one of the robot already contributing to Z as a reference to align the latent space of the newly added robot. The weights ϕ_{ref} are frozen and only ϕ_{new} and θ_{new} , respectively the input and output base weights of the added robot, are adjusted. More formally, the following loss is minimized

$$\min_{\theta_{new}, \phi_{new}} \sum_{(x_{ref}, x_{new}) \in D} L_{B_{new}} + \delta L_{sim} + \lambda L_{CA} \quad (4.7)$$

Once the new robot’s latent space is aligned, it can act as a source or as a target robot for any UNN module training or transfer. This process can be repeated to add as many extra robots as needed.

UNN training

Once the bases training is done, we can proceed to the training of a UNN module $U_{\mathcal{T}\psi}$, approximated by a neural network with weights ψ on a chosen task \mathcal{T} . We parametrize the UNN policy as a Gaussian distribution i.e

$$U_{\mathcal{T}\psi} = \mathcal{N}(\mu_{\psi}(z, o_{\tau}), \Sigma_{\psi}) \quad (4.8)$$

where $\mu_{\psi}(z, o_{\tau})$ is the neural network that maps from observations to mean actions and Σ_{ψ} is the covariance diagonal matrix whose parameters are independent of the state. For an intuitive and visual understanding of how the UNN module fits into the complete policy

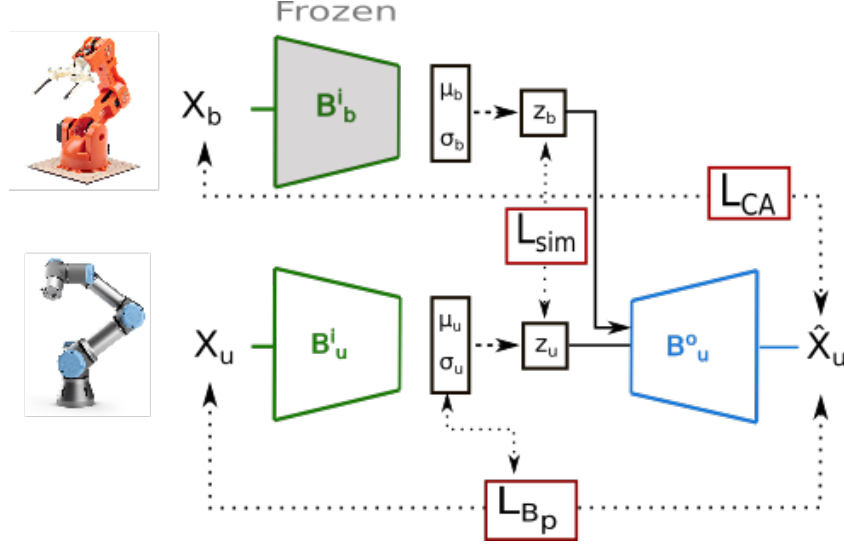
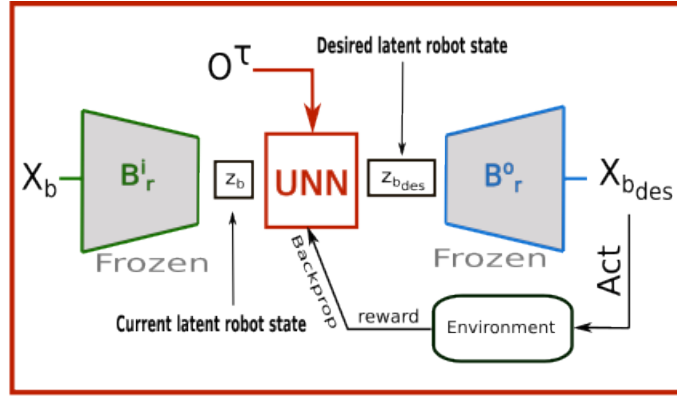


Figure 4.4: Bases fitting process. Above figure describes the methodology to align the latent space of a new robot. The similarity and cross-alignment losses are minimized with respect to a ground truth provided by the frozen input base of a robot already sharing the latent space.



UNN training on source robot braccio

Figure 4.5: Illustration of the UNN training procedure.

architecture, please refer to Figure 4.1. The whole UNN training process is detailed in Algorithm 1. During task training, the bases weights of the robot the UNN is training on are frozen and the gradient is backpropagated through the UNN module only to adjust ψ (see Figure 4.5). From $B_{\phi_r}^i(\cdot|x_r)$ we sample $z \sim \mathcal{N}(z|\mu_{\phi_r}(x_r), \sigma_{\phi_r}^2(x_r))$, input to the UNN module, rather than taking the mean, to keep stochasticity in the observed latent state as a form of domain randomization. We empirically found that it improves UNN robustness against domain shift that may appear when transferring the UNN modules from one robot to another, which in turn improves zero-shot performance after transfer. The deterministic alternative is to take the mean $z = \mu_{\phi}(x_r)$. For the sake of clarity, we rewrite Equations (2.36), the RL objective optimized by the UNN module, with the relevant notations and by using Equation (3.13) and (3.14). Denoting $s_u = (z, o_\tau)$, the concatenation of the current latent state z and the task-specific observation o_τ :

$$L(s_u, z_d, \psi_{old}, \psi) = \min(r(\psi)A^{U\psi_{old}}(s_u, z_d), \text{clip}(r(\psi), 1 - \epsilon, 1 + \epsilon)A^{U\psi_{old}}(s_u, z_d)) \quad (4.9)$$

with

$$r(\psi) = \frac{U_\psi(z_d|s_u)}{U_{\psi_{old}}(z_d|s_u)} \quad (4.10)$$

Since the UNN operates on the shared latent space, it is by definition robot-agnostic and thus can be transferred to any robot of the set by plugging it between the corresponding pair of bases (as shown in step 4 of Figure 4.6).

Algorithm 1: UNN training with PPO

Input: $B_{\phi_r}^i, B_{\theta_r}^o, lr$ (learning rate), N (number of epochs)

Output: Trained U_ψ

```

1 Initialize  $\psi$ 
2 while not converged do
3    $s_r \sim \rho_0$  (initial state distribution)
4    $D \leftarrow \emptyset$  (empty buffer)
5   while  $s_r$  not terminal do
6      $x_r, o_\tau \leftarrow s_r$  (decompose full observed state)
7      $z \sim B_{\phi_r}^i(\cdot|x_r)$ 
8      $z_d \sim U_\psi(\cdot|z, o_\tau)$ 
9      $a_r \leftarrow B_{\theta_r}^o(z_d)$ 
10    Take action  $a_r$  and observe next state  $s_r$  and reward  $r$ 
11     $D \leftarrow D \oplus (z, o_\tau, z_d, r)$  (store transition in buffer)
12  if  $D$  full enough then
13    for  $k = 1..N$  do
14       $z, o_\tau, z_d \leftarrow D$ 
15       $\psi \leftarrow \psi + lr \cdot \nabla_\psi L(z, o_\tau, z_d \psi, \psi_k)$ 

```

You can refer to Figure 4.6 for a visual representation of the full training and transfer pipeline of our approach.

4.3 Experimental setup

In this Section we experimentally demonstrate the effectiveness of our TL framework through 3 increasingly complex robotic tasks on 3 dissimilar robots on simulation. We also provide results for the medium difficulty task on the physical domain with 2 of the considered robots. More specifically we report zero-shot generalization in some instances, where performance after transfer is recovered instantly. In worst case scenarios, performance is retrieved after fine-tuning on the target robot for a fraction of the training cost required to train a policy with similar performance from scratch. We mainly focus our discussion and analysis on the sample-efficiency boost exhibited by our method when compared to regular but state-of-the-art Reinforcement Learning algorithms.

A summary of the robots used and tasks learned is depicted in Figure 4.7. Each possible transfer is studied. The control loop executes at 10 Hz and simulation is run on the Unity physic simulator[Juliani 18]. For the real world experiments, we use ROS [ROS] to operate the robots and the regular joint velocity controllers from the ros-control package.

Robot module transfer

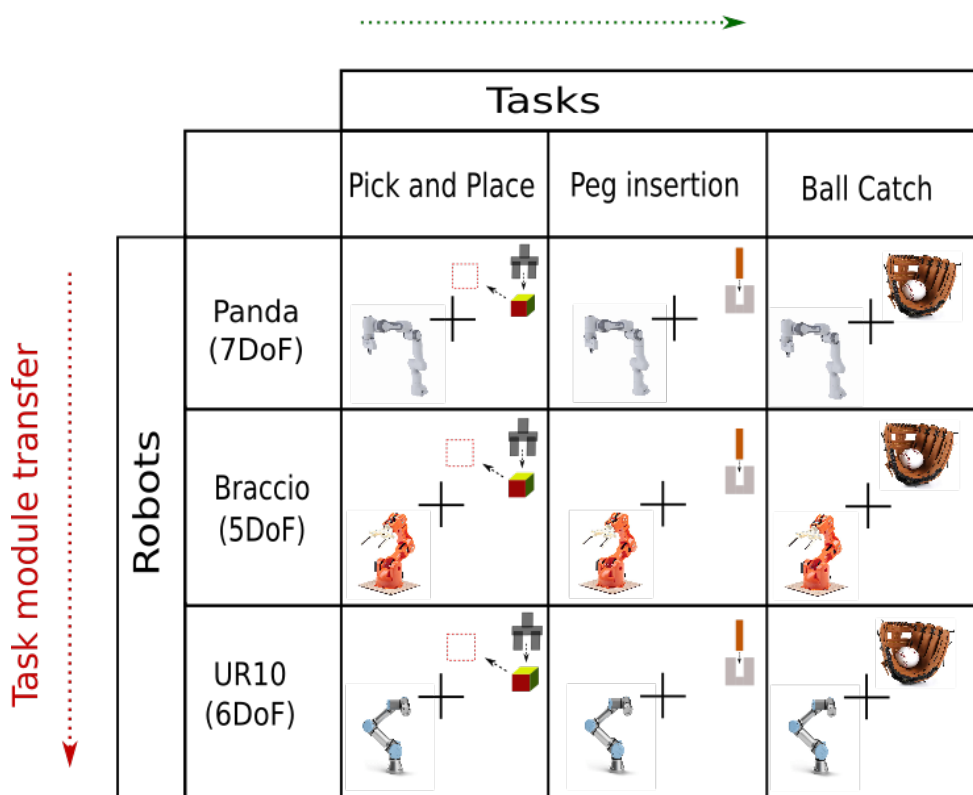


Figure 4.7: Considered Universe. The setup considers 3 robots with different DoF, each of them performing 3 tasks. This amounts to a combination of 9 worlds in total.

robots are velocity controlled, so we ignore the joints position given by the output base when working with the UNN module.

- Pick and Place:** The goal of the agent is to pick a cube from a table and place it at a desired location (see Figure 4.10a). While conceptually simple, it requires the agent to solve a sequential problem: first picking, then placing. It will show that high level understanding of the task and sequencing skills are transferred from one robot to another. The picking part is controlled by a simple boolean set by the agent. If this boolean is true and the suction part of the gripper is colliding with the cube, it sticks. The position of the cube and the desired location are chosen randomly at the beginning of each episode from a range of possible coordinates. The task-related observations for this task are $O^{\mathcal{T}} = \mathbb{R}^{10}$ with current cube position $\in \mathbb{R}^3$, target cube location $\in \mathbb{R}^3$, effector position $\in \mathbb{R}^3$ and a flag *hold* to indicate to the agent if a cube is held or not. The actions applied to the robot are desired joints velocity $\in \mathbb{R}^n$ and a boolean indicating whether the effector should stick or not. The reward function used for training is the following:

$$r_t = \begin{cases} \frac{a}{d_{z,c}} & \text{if } hold = True \\ -b \cdot d_{e,c} & \text{else} \end{cases} \quad (4.11)$$

where $d_{e,c}$ and $d_{z,c}$ are respectively the distance effector-cube and the distance cube-drop zone. The constants a and b are small scaling parameters. In our settings $a = 1.2$ and $b = 0.3$.

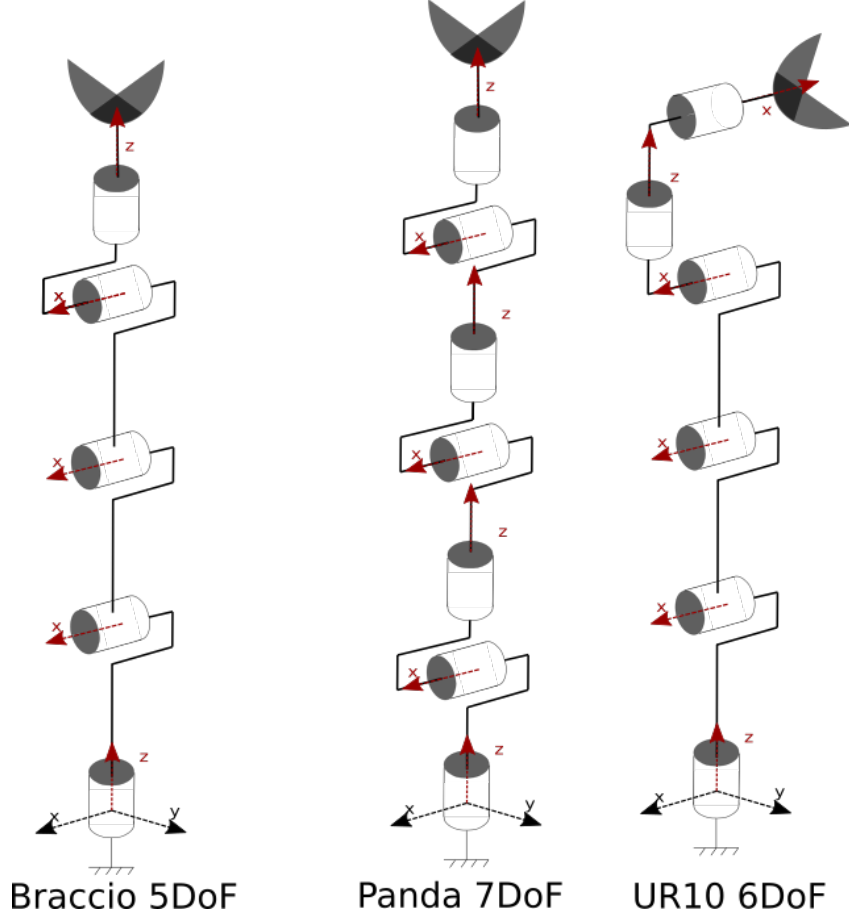


Figure 4.8: Kinematic diagrams of the considered robots

- Peg insertion:** The effector of the robot is replaced by a stick and the agent task is to insert it into a hole of appropriate dimensions (see Figure 4.10c). The position of the hole is randomly sampled at the beginning of each episode from a box region. This task is particularly challenging in the context of cross-robot transfer because it requires a precise control policy. It will highlight the UNN ability to accurately operate the robots it is transferred on, even when their kinematic structure varies significantly. The task-related observations for this task are $O^T = \mathbb{R}^6$ with the tip of the peg position $p \in \mathbb{R}^3$ and the hole position $h \in \mathbb{R}^3$. Like previous task, the actions applied to the robot are desired joints velocity $\in \mathbb{R}^n$. The reward function used for training simply incentivizes the agent to get the peg closer to the bottom of the hole:

$$r_t = \begin{cases} -s \cdot d_{p,h} & \text{if } close = True \\ f & \text{else} \end{cases} \quad (4.12)$$

where $d_{p,h}$ is the distance between the peg tip and the hole bottom. The constant $s = 0.15$ is a small scaling parameter. The boolean *close* is *True* if $d_{p,h}$ is below a fixed threshold and trigger a small positive reward $f = 0.1$.

- Ball Catcher:** The effector of the robot is replaced by a basket and the goal of the agent is to catch a ball thrown at it before it touches the ground. The ball is thrown from the same location but the trajectory is varied by randomly selecting

the ball velocity for each of the 3 spatial axes. This task is used to demonstrate that the UNN is able to cope with dynamic tasks. The task-related observations for this task are $O^T = \mathbb{R}^{13}$ with ball position $\in \mathbb{R}^3$, ball velocity $\in \mathbb{R}^3$, basket position $\in \mathbb{R}^3$ and basket velocity $\in \mathbb{R}^3$ and a flag *catch* to indicate to the agent if the ball is inside the basket. Once again the actions applied to the robot are desired joint velocity $\in \mathbb{R}^n$. The reward function used for training is the following:

$$r_t = \begin{cases} -c \cdot d_{b,t} - \beta |\theta_e| & \text{if } catch = False \\ e - \beta |\theta_e| & \text{else} \end{cases} \quad (4.13)$$

where $d_{b,t}$ is the distance between the basket and the target receiving position. The constant c is a small scaling parameter and e is a small positive constant reward. In our settings, $c = 0.15$ and $e = 0.15$. The term $|\theta_e|$ is the angle between the effector pose and the vertical plane which guides the agents towards suitable body configurations. Its contribution to the overall reward is weighted by the constant β . Interestingly, this reward term is required by the regular RL agents to learn a successful policy but not for our UNN agents, already imbued with prior knowledge about adequate effector orientation through the bases.

To summarize, we perform transfer between 3 different robot morphologies with 3 tasks which demonstrate the ability of our transfer method to cope with: planning tasks, precision tasks and dynamic tasks.

For the real world experiments, we transfer the UNNs and the RL agents corresponding to the peg insertion task trained in simulation, to the physical Panda and UR10 only. The physical Braccio robot was not reliable enough to attempt a peg insertion task due to significant mechanical play and offsets in the joints. The position of the peg hole is measured in real time using a motion capture system from qualisys. The position of the peg tip is computed using the Forward Kinematic provided by the robot’s manufacturer. The robots and the task setup can be seen in Figure 4.9.

4.3.3 Modules Training

In this part we go over the actual implementation details of our methodology to learn both the robot-specific modules and the task modules. We describe all the relevant technical and training details used for the results displayed in section 4.4.

Bases Training

We arbitrarily chose to use the Braccio-Panda pair for the latent space creation but any pair of robots could have been used. Future work may investigate the impact of such a choice on downstream transfer success. Following the training of the Braccio-Panda modules, we aligned the bases of the UR10 robot using the Braccio robot input module as reference. The **primitive task** used for the bases training was a simple **reaching task**. A target is randomly moving in the robot work-space and the goal of the agent is to put the robot’s effector at the desired pose (see Figure 4.11 for an illustration). Although it is a very basic task, it is the fundamental building block of a lot of more advanced robotic skills. Therefore, we hypothesized that most robotic behaviors could be recovered from a large enough number of reaching trajectories, yielding a suitable latent state-action space for the UNN.



(a) UR10 robot.



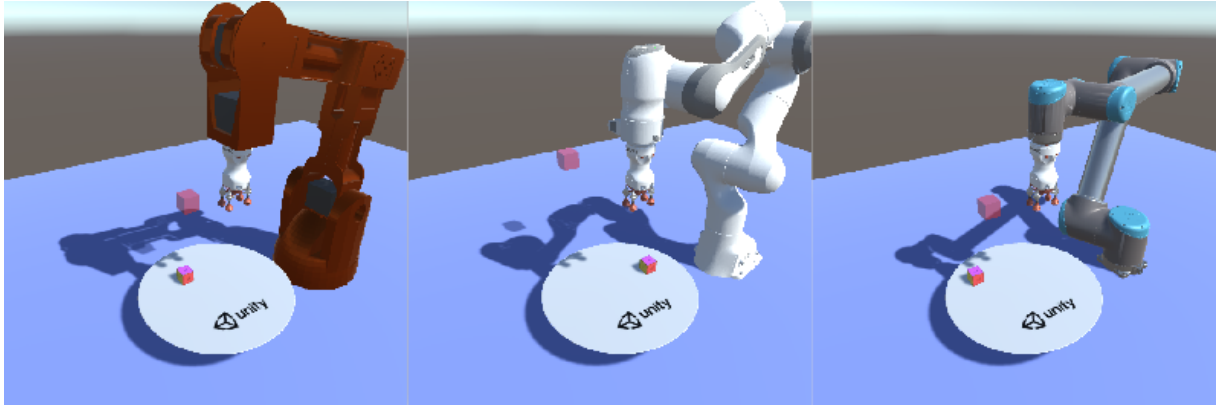
(b) Panda robot.

Figure 4.9: Peg insertion task setting with the physical robots.

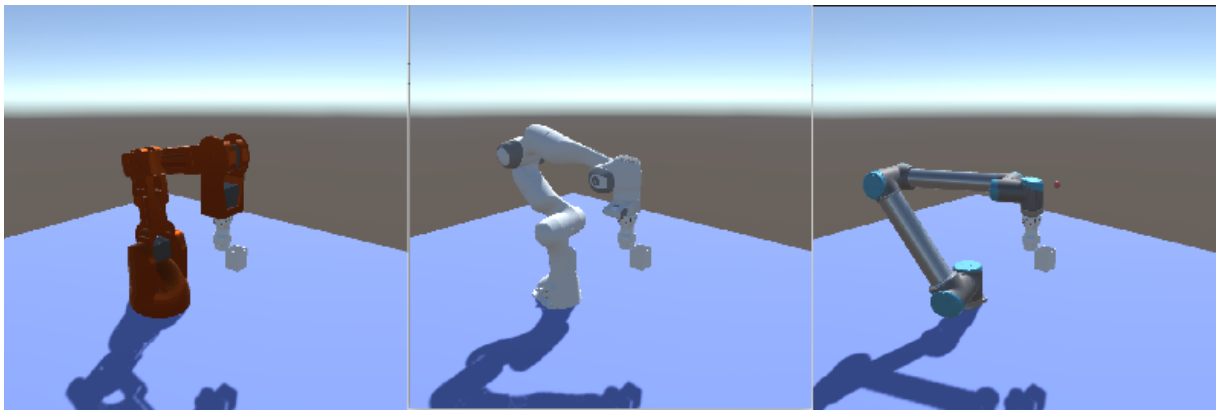
The optimal policies for the primitive task were obtained with analytical methods [Starke 17]. More specifically, we simply provide the target position to a general inverse kinematic solver. As the bases represent the robots from the UNN’s point of view, it is possible to include a priori knowledge of how the robot should behave. For example, the tasks that the robots need to achieve require the effector perpendicular to the horizontal plane, so we constrained our inverse kinematic solutions accordingly. As a consequence, the output bases will mostly generate trajectories suited for the given tasks.

We determined the optimal size of the latent space based on the reconstruction error for a fixed β , the weighting constant for the KL-divergence in Equation (4.3) (see Figure 4.12). The reconstruction error is a good metric in this case because it measures how much information about the input data is contained in the latent variables. Naturally, a low dimensional bottleneck (a dimension of 2 for instance) has not enough capacity to encode all the required information for proper reconstruction by the decoder. On the contrary, increasing the latent space size too much results in over-fitting as the network starts memorizing the data rather than finding a robust and meaningful low-dimensional representation. As a result, the reconstruction error on the test set increases. As shown in Figure 4.12, a latent space of dimension 6 achieves the best results for an input size of 10 (Braccio) and 14 (Panda).

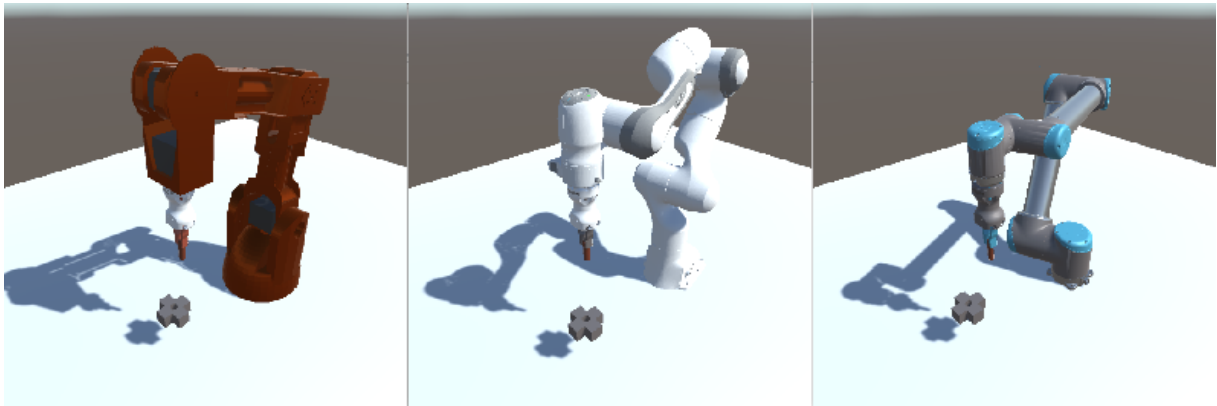
The recorded trajectories ultimately define the latent space so it is paramount to ensure that we visit a large variety of states in the work space to obtain a sufficiently rich latent space. We chose to record a dataset consisting of 100 000 pairs of similar states. In order to get them, the collected datasets were time-aligned as discussed in Section 4.2.1.



(a) Pick and Place task. The picking cube is placed on the table and the goal position is represented by the red transparent cube.



(b) Ball catcher task. The ball is thrown from a fixed location in front of the robot.



(c) Peg insertion task. The grey platform contains the square hole and the robot clamp holds the cylindrical peg.

Figure 4.10: Considered tasks for the experiments. From top to bottom: Pick'n Place, Ball catcher and Peg insertion. From left to right: Braccio robot, Panda robot and UR10 robot.

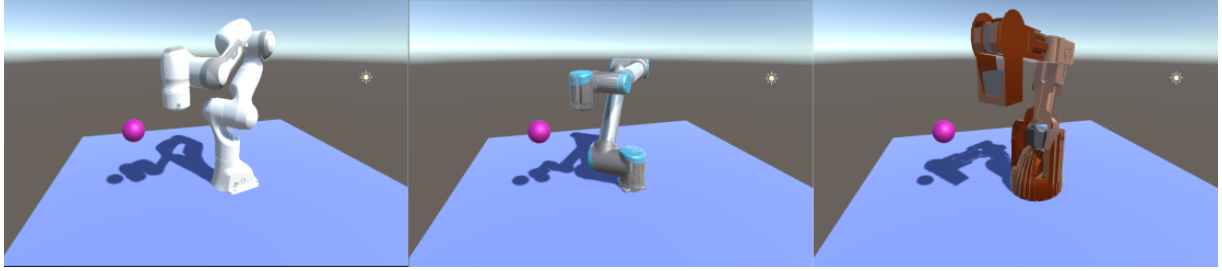


Figure 4.11: Primitive reaching task. The violet reaching target can move freely inside a reachable pre-defined working space.

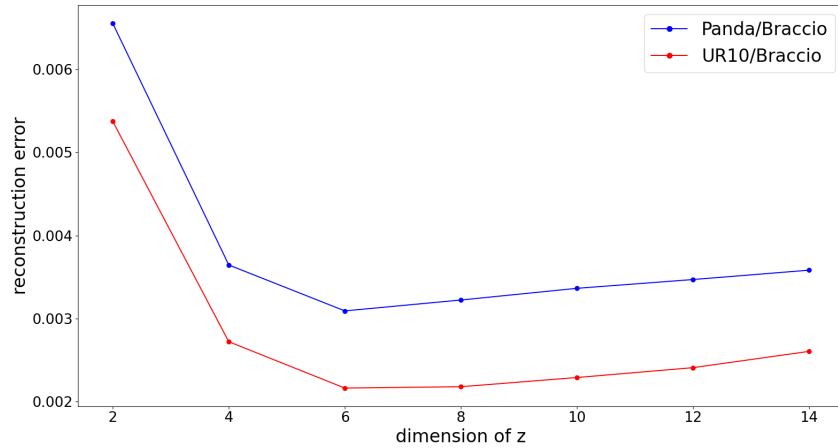


Figure 4.12: Averaged reconstruction error (on the test set) as a function of the dimension of the latent space Z , for the bases training. Both robot pairs are tested.

To do so, the target to be reached was moving at constant speed and visiting the same locations during the primitive task execution, for all robots considered. The robot state (joints position and velocity) was sampled at 40 Hz. In total, the trajectory recording took approximately 45 minutes per robot. As for the bases training, it took around 15 to 20 mins to complete on a single CPU without GPU parallelization. Figure 4.13 depicts the plotted latent space after applying principal component analysis (PCA) for dimension reduction (μ components are the mean of the gaussian distributions outputted by the input base). It can be seen, the latent space of both robots are well aligned and regularized (clustered around zero, see Figure 4.14). Figure 4.14 is also interesting because it indicates the span of the latent space. Using this plot, we can deduce the spread of each latent variable to scale the UNN action space accordingly.

The input and output bases were both approximated by a neural network of 3 hidden layers and leaky ReLU with a negative slope of 0.01 were used as activation functions (see Section 2.3.3 for a recall). The full neural network architecture is detailed in Figure 4.15. For the weighting constant of equation A.8, we chose $\delta = 2/3$, $\lambda = 1/3$ and $\beta = 0.00015$. We trained for 100 epochs with a learning rate of $5 \cdot 10^{-4}$ and batch size of 100.

UNN and baseline agents trainings

We use the PPO implementation of ml-agents [Juliani 18] modified to suit our needs for both the UNN task modules and the RL agents training. Their feed-forward neural

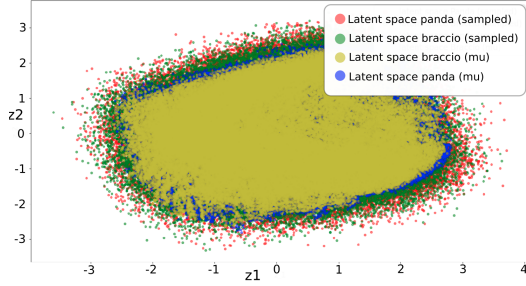


Figure 4.13: Latent space for the Braccio/Panda pair after applying PCA for dimensionality reduction and visualization on a 2D space.

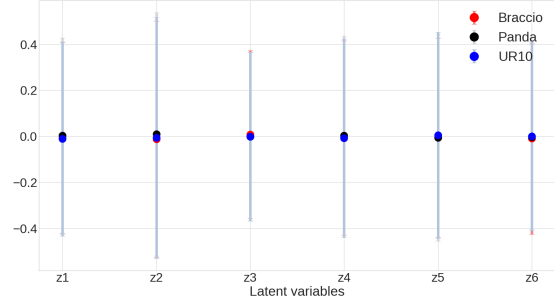


Figure 4.14: Mean and standard deviation for each latent variable. The mean and standard deviation of the latent variables are the same across robots.

networks is composed of 3 hidden layers with 225 neurons each (see Figure 4.15 and Figure 4.16 for a full description of the neural network structure). The discount parameter γ is 0.99 and λ (GAE) is set to 0.95. An entropy bonus term of weight 1.10^{-2} , linearly decayed over time, is added to the reward to incentivize exploration in the early stage of training. The learning rate used is 3.10^{-4} and the clipping parameter ϵ is 0.2. Both hyper parameters were also linearly decayed to improve learning stability. We used a buffer of size 40960 with a batch of size 2048. The number of epochs for each policy update was set to 6. The inputs provided to the agents as observation are first normalized by tracking a running mean and variance. Finally, the non linear activation function used is the SiLU [Ramachandran 17]. The choice for the activation function is motivated by the fact that SiLU has been empirically proven to significantly outperform ReLUs on some RL environments [Elfwing 18].

As it is often the case when working with continuous action spaces, we clip the output of the network between -1 and 1 and scale it as needed. As explained earlier, we need to constraint the UNN action space during training to ensure that it does not wander outside of the latent space known to the decoder/output base. As such, it is critical to choose the scaling factor of the UNN clipped output carefully. At the end of the bases training, we compute the standard deviation σ of the latent variables to the extent of the latent space is. Then, we set the scaling factor to 3σ to ensure that the UNN has access to the full latent space, but cannot operate outside of its defined boundaries.

4.4 Results

In this section, we present our results both on training and transferring on the chosen manipulation tasks. The main goal of these experiments is to demonstrate the effectiveness of task transfers using LS-UNN and as such, the training time saved when compared to regular RL training. Videos demonstrating some of our experiments are available here. Code can be found at <https://github.com/sabeaussan/LS-UNN>. Inspired by early work on TL for RL agents [Taylor 07], we analyze three metrics when comparing both methods:

- **Zero-shot performance:** Initial performance of the UNN agent right after transfer. It will measure how useful the knowledge transfer is to jumpstart learning on the target robot.

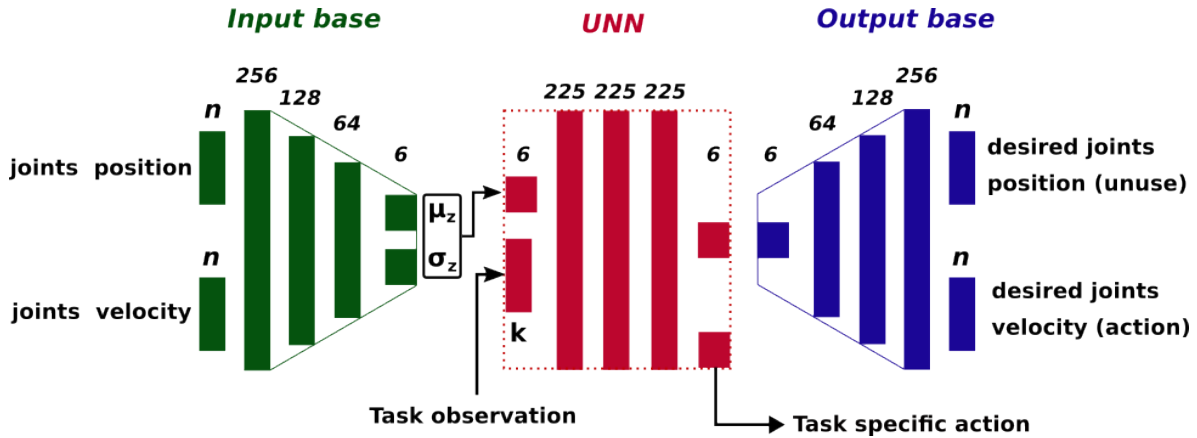


Figure 4.15: Neural network architecture of the full UNN network. In this figure, we set the latent space to a size 6 and suppose a task with observation of dimension k performed by a robot with n DoF. In our experiment we only make use of the joints velocity given by the output base. The task specific action is not processed by the output base (e.g suction boolean for the pick and place task).

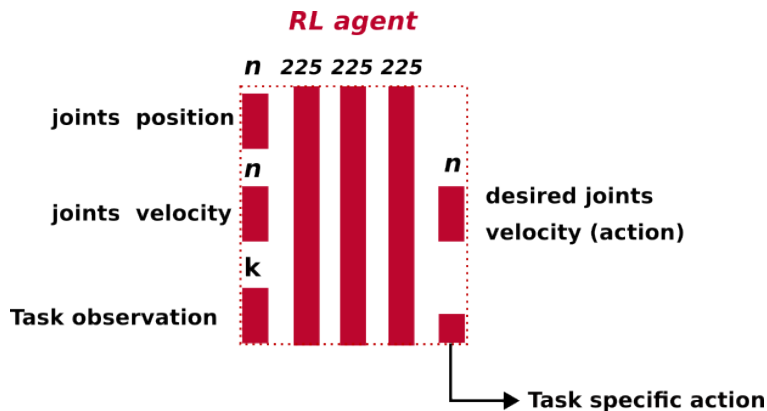


Figure 4.16: Neural network architecture of the full PPO agents. We suppose a task of dimension k performed by a robot with n DoF.

- **Asymptotic performance:** The final performance obtained after training from the Zero-shot performance of the UNN agent (fine-tuning) or from scratch. This will show whether or not our method falls into local minima or enables similar asymptotic performance as baselines.
- **Sample efficiency:** Number of samples needed for the agent to reach its final performance. This is, arguably, the most appropriate metric for evaluating a transfer learning method, especially in RL which is notoriously sample inefficient.

The aforementioned metrics reflect our primary concern: to improve the learning speed of RL-trained agents, i.e how much interaction with the environment is needed for an agent to learn a high-performance policy. In this regard, our baselines will consist of agents trained from scratch with regular PPO on the target robot. They will serve as a upper bound for the performance and corresponding amount of training steps needed when using state-of-the-art RL algorithms. In contrast, our method also leverages PPO

to fit the UNN module, but inside a pre-trained and shared latent space.

Our main focus is zero-shot transfer as it is the most appealing kind of transfer, since it requires no further training on the target robot. It is as sample efficient as can be. We therefore compare zero-shot performance of the transferred UNN to the asymptotic performance of the corresponding baseline to emphasize the training-time saved. However, in some instances, source performances cannot be recovered directly after transferring the UNN agent. To correct this, the UNN module can be fine-tuned on the target robot to recover near maximum performance. It is realized with a fraction of the training time needed by the pure RL baselines. Finally, it is demonstrated that the prior knowledge included in the bases can, in some cases, speed up the learning of the tasks on the source robot (i.e the UNN training), when compared to standard PPO training.

We train one UNN module per robot for each task (i.e 9 UNN modules in total) before transferring it to the other robots of the pool. As explained earlier, when varying the task, the robot bases are kept the same. We then measure performance as the task success rate, i.e the percentage of successful episodes over 1000 trials. For the pick and place task, an episode is considered successful if the distance between the cube and the place location is smaller than a threshold set to 0.15 Unity units (approximately 3 cm for an environment scaled for the UR10 robot). For the ball catching task, we define success as simply catching the ball, i.e the ball does not bounce outwards and is kept inside the basket. In the case of the peg insertion task, we also define task success with a distance threshold set to 0.05 Unity units or 1 cm between the peg tip and its desired position inside the hole. For the execution on the real robot the success threshold is set according to the scale of the robot. Regarding the experiments on the physical robots, we test 28 different positions uniformly spread out on the working area. Its size is scaled proportionally to the robot total length (approximately 1 cm). The square hole (depicted in Figure 4.17) is 2.25 cm x 2.25 cm wide and the peg diameter is 1.9 cm. The bases used on the physical robots are the same as the ones used on their virtual counterparts.



Figure 4.17: Close-up look on the red square hole and blue peg used for the peg insertion task. The square hole is 2.25 cm x 2.25 cm wide and the peg diameter is 1.9 cm

4.4.1 Zero-shot transfers

Here we discuss the performance obtained on the target robot right after transfer from a different source robot, without fine tuning the UNN module. In particular we compare immediate performance on the target robot after transfer to the performance of the corresponding RL baseline which serves as an upper bound. In other words, we answer the question "What kind of performance can be expected with a simple plug-and-play of the UNN module on a target robot?". As we are ultimately interested in reducing the training time on the target robot, it could be interesting to see if a UNN transfer recovers performance close to the source robot or even RL baseline only with zero-shot transfer.

Pick and place

On the pick and place task, which deals with high-level task-sequencing, nearly all transfers are zero-shot. Except for Braccio \rightarrow Panda transfer, maximum performance is recovered after direct transfer without the need for further training on the target robot as shown in Table 4.1. It means that we get results competitive with those of the corresponding RL baselines, without any fine tuning or training on the target robot, saving us an average of 1.8 millions training steps required to reach convergence (see Figure 4.18). As such, results indicate that the high-level understanding of the task is transferred along with the UNN module.

Source \ Target	Braccio	Panda	UR10
Braccio	100	93 / 100	100
Panda	100	100	100
UR10	100	100	100

(a) Performance obtained for UNN agents (zero-shot/**fine-tuned**).

Braccio	Panda	UR10
100	100	100

(b) Performance obtained for RL agents.

Table 4.1: Pick and Place task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.

Ball catcher

Regarding the zero-shot performance on the ball catcher task shown in Table 4.2, we can see that half of the transfers exhibits zero-shot performance close to the RL baselines performance: Braccio \rightarrow UR10 (98%), Braccio \rightarrow Panda (98%) and UR10 \rightarrow Panda (98%). However, some of the zero-shot transfers experience a performance drop: Panda \rightarrow Braccio (86%), Panda \rightarrow UR10 (90%) and UR10 \rightarrow Braccio (89%). This can be explained by the fact this task requires much more precise movements to catch the ball. If, for instance, the basket is tilted too much on the left or is leaning a little too far forward, the ball will bounce back outward, hit the basket's edge or miss it, resulting in task failure. The reconstruction error occurring during the bases training will inevitably induce these small parasitic movements. As a consequence, this task is less tolerant towards imprecision and more challenging when it comes to transfer. Moreover, results seem to suggest that UNN policies learned on higher DoF robots tend to not transfer

well to robots with less DoF. All transfers from the Panda robot (7 DoF) need further fine tuning on the target robot. The same phenomenon can be observed from UR10 (6 DoF) to Braccio (5 DoF). We hypothesize that the UNN will somehow take advantage of the extra joints when learning its policy, yielding a policy that may not be adapted to less "expressive" robots. Though, performance is still high enough to demonstrate that transfer is beneficial and as it will be presented in section 4.4.2, maximum performance can be recovered with a fraction of the required training for RL baselines.

Source\Target	Braccio	Panda	UR10
Braccio	98	98	98
Panda	86 / 98	98	90 / 98
UR10	89 / 98	98	99

(a) Performance obtained for UNN agents (zero-shot/**fine-tuned**).

Braccio	Panda	UR10
100	99	99

(b) Performance obtained for RL agents.

Table 4.2: Ball Catcher task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.

Peg insertion

Simulation results: This task is also very challenging for transfer between robot’s morphologies as it requires accurate movements and careful control of the robot to successfully insert the peg. Nevertheless, success rates right after transfer of the UNNs are significantly high, indicating a beneficial and efficient transfer of knowledge between the considered agents as shown in Table 4.3. But once again, fine tuning is required to recover from performance loss after transfer.

Physical results: We also show results from transferring the UNN policies on the physical UR10 and Panda robots. The UNN were trained in simulation and the bases used are the same as their virtual counterparts. The fine-tuned UNNs are also the ones obtained in simulation, they are not fine-tuned on the physical robots. Results are reported in Table 4.4. All the considered agents experience a performance drop after the sim2real transfer. In particular UR10 \rightarrow Panda decreases from 90% to 71% success rate in zero-shot. Likewise, Braccio \rightarrow UR10 suffers from the sim2real transfers but its fine-tuned version performs well. Despite the large reality gap, performance on the physical robots is still above 85% after fine-tuning on the virtual robots.

4.4.2 UNN fine tuning

In this section, we analyze the performance of the transferred UNN after fitting it on the target robot. When fine-tuning the UNN agent on a target robot, we remove stochasticity in the observed latent state (see Section 4.2.2), i.e $z = \mu_{\phi_r}(x_r)$. We are especially interested in the time gained by simply fine tuning the UNN module as opposed to training a RL agent from scratch. As such we will discuss the Asymptotic performance and the sample efficiency metrics described in Section 4.4. As shown in Figures 4.18, 4.19 and 4.20, most of the fine tuned UNN agents start from an already high jumpstart performance and

Source\Target	Braccio	Panda	UR10
Braccio	100	95 / 100	92 / 100
Panda	95 / 100	100	82 / 100
UR10	99 / 100	90 / 100	100

(a) Performance obtained for UNN agents (zero-shot/**fine-tuned**).

Braccio	Panda	UR10
100	100	100

(b) Performance obtained for RL agents.

Table 4.3: Peg Insertion task: performances on the simulated robots. Results are reported as percentage of task success over 1000 trials.

Source\Target	Panda (physical)	UR10 (physical)
Braccio (virtual)	79 / 93	75 / 93
Panda (virtual)	93	79 / 96
UR10 (virtual)	71 / 85	96

(a) Performance obtained for UNN agents (zero-shot/**fine-tuned**).

Panda	UR10
89	100

(b) Performance obtained for RL agents.

Table 4.4: Peg Insertion task: performances on the physical robots. Results are reported as percentage of task success over 28 trials.

converge faster to the highest performance than their RL counterparts. It is worth noting that PPO is a trust-region based RL algorithm which theoretically provides monotonous improvement by solving a KL-constraint. As such it is well suited for fine tuning because it mitigates catastrophic forgetting and improves training stability.

Pick and Place

For the pick and place task (see Figure 4.18), most of the transfers are already close to baseline so fine tuning was performed only for the Braccio \rightarrow Panda transfer. While the UNN has experienced a performance drop, it recovers competitive results significantly faster than what was needed for its RL counterpart to reach the same performance. More precisely, it achieves roughly a 2x speed up over a regular PPO training.

Ball Catcher

For the more dynamic ball catcher task, fine tuning is necessary on half of the transfer. However, all considered transfers achieve competitive performance provided some quick fine tuning. The Panda \rightarrow Braccio and UR10 \rightarrow Braccio transfers which had the lowest zero-shot performance (86% and 89%), reach near baseline performance in respectively 1.6 millions and 2 millions steps of fine-tuning against the 4.5 millions required by the regular RL agent, yielding a 2.8x and 2x speed up. Regarding the UR10 \rightarrow Braccio transfer starting with a 90% performance, the fine-tuning required only 14% of the amount of training samples needed by its corresponding RL baseline to achieve similar performance.

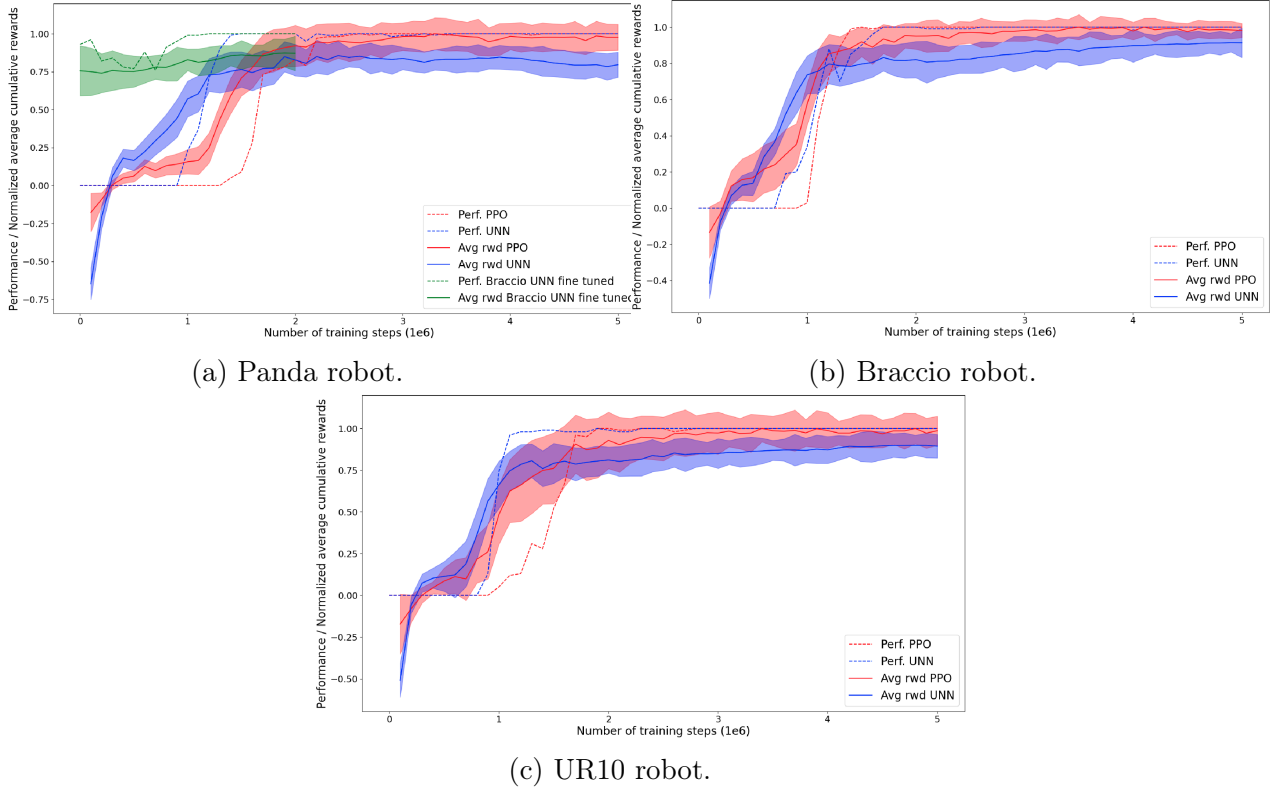


Figure 4.18: Pick and place task. Training curves on considered robots: y-axis is the normalized average reward / average performance and x-axis is the number of training steps. The PPO agent refer to our baseline, the UNN agents refer to the transferable task module.

Peg Insertion

None of the transfers retrieve full performance after zero-shot transfer. Nevertheless, success rates are already close to baseline. Braccio \rightarrow UR10 has the lowest zero-shot success rate amongst all transfers, successfully achieving the task 82% of the time. Yet, Figure 4.20c shows that the Braccio UNN can quickly adapt to the UR10, reaching maximum performance with as little as 2% (100000 steps against 3.8 millions) of the experience required by the corresponding baseline. Other fine-tuned transfers depicted in Figure 4.20 exhibit similar recovery speed and improved sample efficiency.

4.4.3 Agent’s training

In this section we answer the question: *"Are there any benefits or drawbacks in learning a task with the UNN module?"*. To this end, we analyze and compare the convergence speed and asymptotic performance reached by the UNN agents during training against vanilla RL baselines. Transfer performances have been already discussed, so we focus solely on training performance. We train the agents for $6 \cdot 10^6$ steps on the pick and place task, $7 \cdot 10^6$ steps on the ball catching task and $6 \cdot 10^6$ steps on the peg insertion task. We monitor their improvement through the average cumulative reward obtained per episode. This metric measures how well the agent behaves within its environment and as such informs us of which approach yields the better results, learning-performances wise.

As shown in Figures 4.18, 4.19 and 4.20, except for the Braccio robot on the pick and

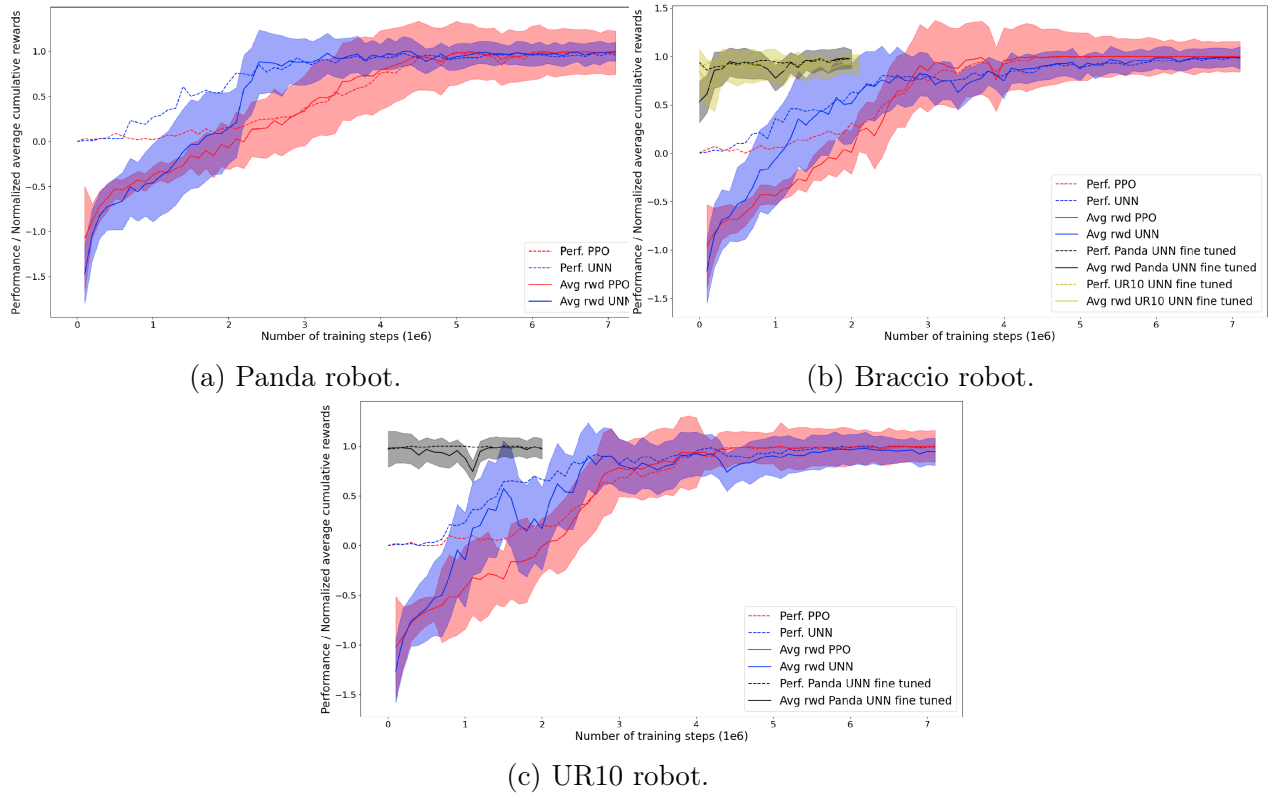


Figure 4.19: Ball catcher task. Training curves on considered robots: y-axis is the normalized average reward / average performance and x-axis is the number of training steps. The PPO agent refers to our baseline, the UNN agents refers the transferable task module.

place and ball catcher tasks, the UNN agents trained from scratch slightly outperform the Vanilla RL agents in terms of convergence speed. As explained in section 4.2.1, during state collection the robots effector orientation is constrained such that they remain vertical and these desirable body configurations were used to train the bases. Consequently, we believe that the latent space embeds states that are well suited for the tasks, in contrast to the original state-action space containing all kinds of inappropriate joints configurations. As such, exploration is faster and more efficient within a compressed latent space.

However, despite similar asymptotic performance on the pick and place and peg insertion tasks (depicted as dashed curves), UNN agents systematically converge to lower asymptotic average cumulative rewards. We hypothesize that it is due to the nature of the latent space. It is a compressed representation of the state-action space of each robot constructed from a finite number of trajectories. As such, it may not include the optimal policies reached by RL agents, which results in slower and sub-optimal trajectories for task execution. Hence, as the considered reward functions penalize slow behaviors, UNN agents will end up with a lower reward even though they complete the task. This is not the case in the ball catcher task, because the ball cannot be caught faster or slower than its throwing speed.

Overall, these results seem to indicate that a UNN agent trained from scratch will learn slightly faster than the RL agent on the same robot, which may be attributed to the prior knowledge included in the robot’s bases. However, they may be unable to obtain similar rewards depending on the task and reward function shape, which indicates a less optimal policy with respect to the RL objective. Furthermore, it also worth taking into account the extra time needed to train the bases when assessing the training cost saved.

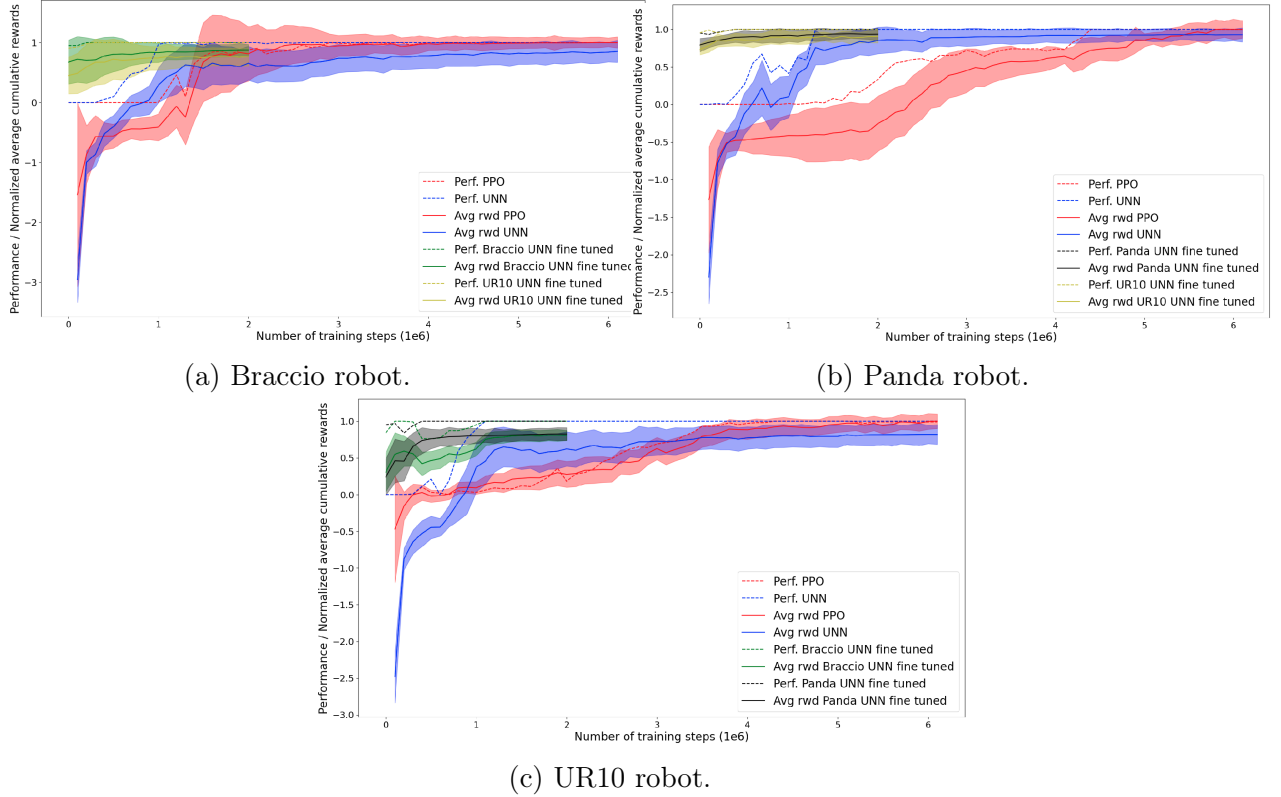


Figure 4.20: Peg insertion task. Training curves on considered robots: y-axis is the normalized average reward / average performance and x-axis is the number of training steps. The PPO agent refer to our baseline, the UNN agents refer to the transferable task module.

4.4.4 Over-fitting

In essence, our methodology promotes robot-agnosticity of the UNN module, by training within a latent space that is common among the robots considered for transfer. By doing so, our goal is to develop a module usable consistently by our pool of robots, regardless of their physical characteristics. However, in practice, we observe significant over-fitting when learning a UNN module depending on the robot platform. It clearly appears when we monitor the performance of the UNN module on the source robot (training domain) and its corresponding performance on the target robots (testing domains) at several points during training. Figure 4.21 depicts the performance curves of the UNN trained on each possible source robots, and evaluated on the rest of the available robots, for the peg insertion task. As shown, the UNN performance is relatively homogeneous at the beginning of training both on source and target robots. However, after the UNN has converged on the source robot, performance starts to degrade on the test robots in most cases (except in Figure 4.21c with the Braccio robot). It seems to indicate that the UNN module finds and exploits robot-specific strategies when trying to keep increasing and optimizing the RL objective.

As a workaround we checkpoint the UNN models every 50000 training steps and take the best performing one. All previously presented results were obtained using this simple method. Moreover, as mentioned in section 4.2.2, we empirically found that sampling z from $B_{\phi_r}^i(\cdot|x_r) = \mathcal{N}(\mu_{\phi_r}(x_r), \Sigma_{\phi_r}(x_r))$ instead of taking $z = \mu_{\phi_r}(x_r)$ improves overall transfers by acting as a form of domain randomization. We let the design of a more

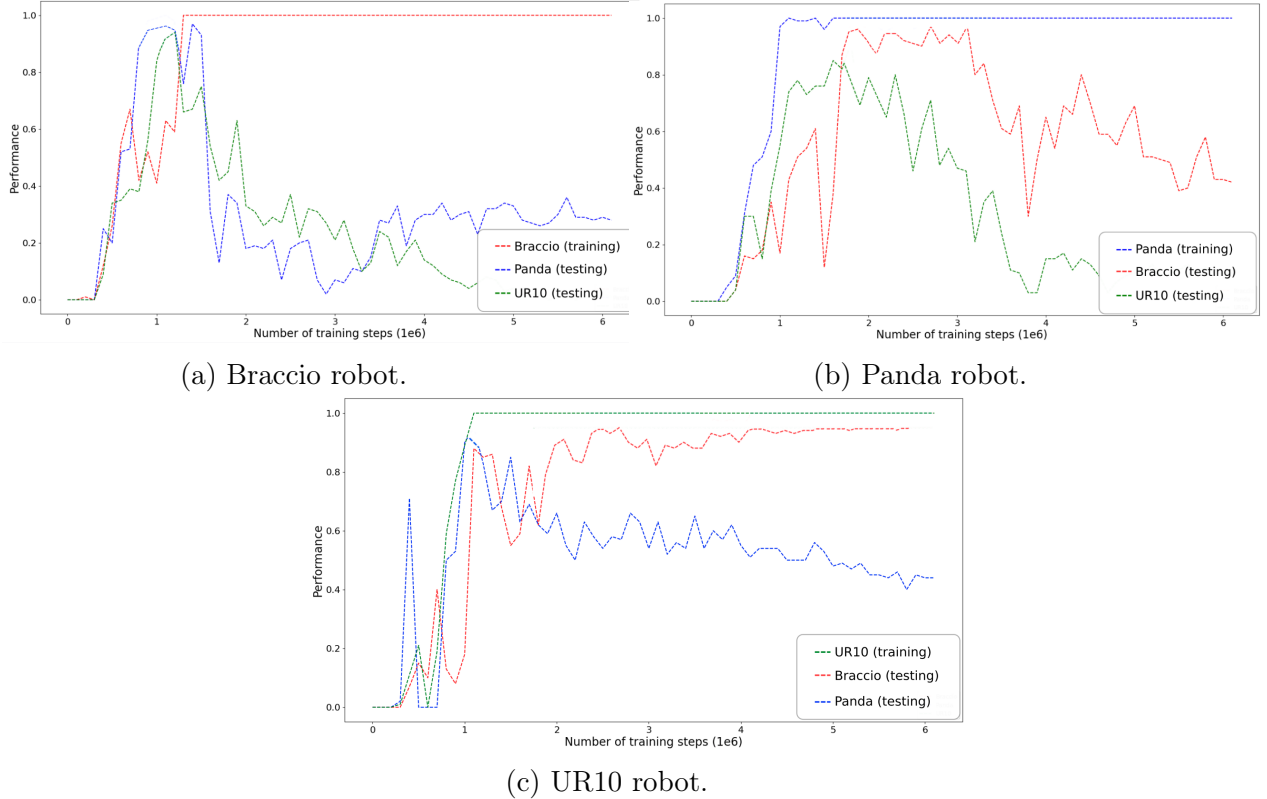


Figure 4.21: Peg insertion task. Performance curves on considered robots: y-axis is the performance and x-axis is the number of training steps. The UNN agent was trained on the (a) Braccio robot, (b) Panda robot, (c) UR10 robot, and tested on the rest of the available robots. Performance on the test robots is recorded every 50000 steps.

robust and sophisticated method to future works.

4.5 Relation to prior works

Cross-Agent Transfer is concerned with reusing and leveraging knowledge and past experience from a morphologically distinct agent in order to speed up the learning of a target agent. It is a very promising avenue of research in the quest for mitigating the notorious sample-inefficiency of RL. Despite its appealing practical application, it is still an understudied sub-field of transfer learning and only a handful of prior works has tackled the cross-agent transfer learning problem. In this section, we review relevant works on the topic of transfer learning in robotics. For the sake of clarity we propose to classify the methods discussed in three broad categories that we introduce in the next sub-sections. We analyze how our work fits into that body of literature and draw connections to other related fields such as domain adaptation, meta-learning and hierarchical reinforcement learning.

4.5.1 Learning a robot-agnostic policy

In this section, we present methods that achieve Cross-Agent Transfer learning by training a single policy on a large variety of robot morphologies, which can be thought as a form

of domain randomization. Their main goal is to learn an agent-agnostic policy, that is, a policy that can control a wide range of robots, regardless of their shapes.

Hardware Conditioned Policies (HCP) from [Chen 18] trains a single policy on a variety of body configurations in order to achieve agent-agnostic control. However, their main contribution to earlier methods is to condition the policy both on the state and on a vector representing the robot hardware. This vector can be as simple as an explicit description of the robot kinematic following the popular URDF format, or learned to accommodate with complex locomotion tasks requiring the policy to have knowledge about dynamic factors such as friction or damping in motors. To account for differences between state-action space dimensions, they used zero-padding in the input and output of the policy network. Their method successfully learned a policy able to control 9 different variations of the same robot, including modification on the number of degrees of freedom and link length (see Figure 4.22). They also showed interesting zero-shot results on a Peg Insertion task. Nevertheless, reported performances are lower than ours on the most similar setting: the transfer to a robot with a different number of DoF only yields 23% success rate.

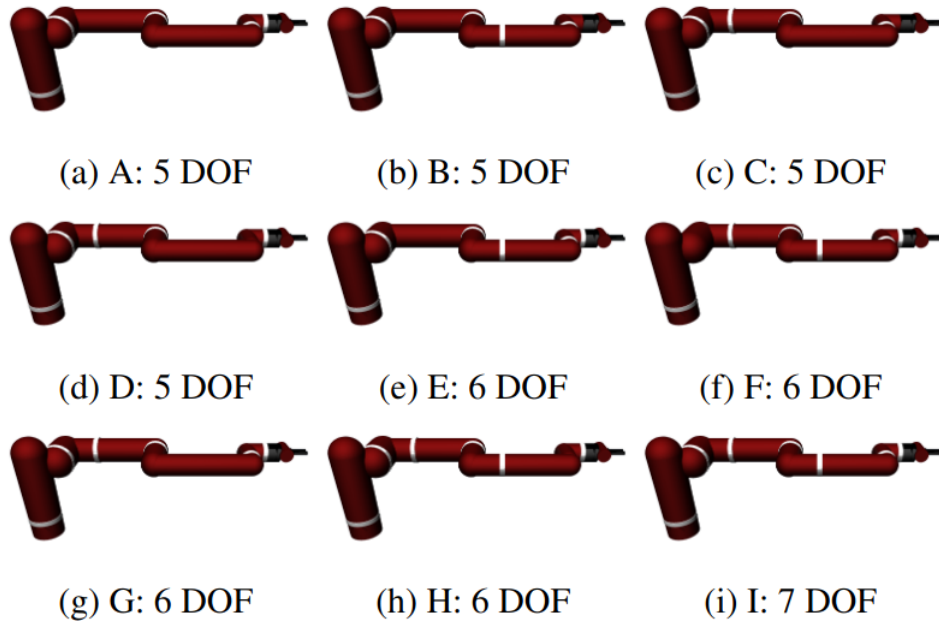


Figure 4.22: Variations based on the Sawyer robot in MuJoCo. Joints are represented by white rings and are shuffled to create different kinematic chains [Chen 18].

An effective and meaningful extension to their work is "Hardware Agnostic Reinforcement Learning" (HARL) [Jackson 21]. More specifically, they added an adversarial network responsible for sampling morphologies for the policy network to train on, using a "learning potential" defined as the ratio between performance in the environment following the policy and performance in the environment following the same policy after K updates. Additionally, they introduce a modification network which takes as input the hardware vector and the action given by a pre-trained expert network conditioned on the environment state, and outputs the modified action suited for the specific robot-morphology. They report improved results both on learning speed and zero-shot performance when compared to HCP. However, their approach was only tested on simple 2D tasks and mor-

phologies such as cartpole and BipedalWalker from the open-AI gym [Brockman 16] (see Figure 4.23. Furthermore, they did not consider transfer between agents with varying DoF for their experiments, keeping the morphologies tested very similar in contrast to previous work and our contribution.

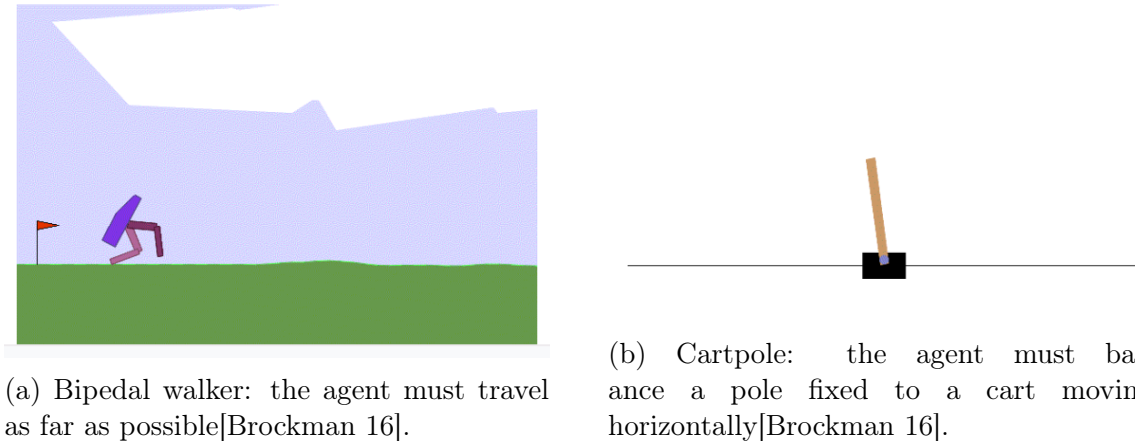


Figure 4.23: Samples of environments used for the experiments on HARL. Multiple variations of these robots were designed by changing links length.

A different but related approach [Ghadirzadeh 21], frames the challenging problem of hardware-agnostic policy training as a few-shot learning problem. In the general case, this type of problem consists in learning to adapt to a new, but related situation using as little data as possible. It is usually addressed and solved using Meta-learning, a sub-field of Machine Learning which focuses on learning to learn. In contrast to traditional ML, meta-learning takes a higher-level perspective and aims to improve the learning process itself. It involves training models on multiple related tasks or datasets to acquire general knowledge or "meta-knowledge" that can be applied to new tasks with minimal additional training. In other words, instead of directly optimizing for performance on a single task, they optimize for learning strategies or initialization parameters that enable rapid adaptation to new tasks. Regarding the approach described in [Ghadirzadeh 21], the goal is to adapt an action-selection policy to a new robot from small amounts of new data. They address this problem using a gradient-based meta-learning method [Finn 17] in order to model a common structure shared across different robotic platforms to enable fast and sample-efficient adaptation. Furthermore, like previous works and the UNN method, they decompose their robot-agnostic policy into a high-level robot-agnostic network generating action latent variables, and low-level robot-specific generative models to produce motor action trajectories. Their results, obtained on 400 variations of 4 types of 7 DoF robots (see Figure 4.24 below), for a reaching and a pick and place tasks, show that the trained policy is able to adapt to unseen robotic hardware in a few-shot manner. Unfortunately, their method was not designed to adapt policies for robots with different number of DoF.

A recent but promising line of work, is to explicitly model the structure of an agent as a graph where limbs and actuators can be represented as edges and nodes (see Figure 4.25 for an illustration). Such a representation, when used with the appropriate kind of neural networks, is more flexible, transferable and generalizable [Wang 18]. For instance, Graph Neural Networks (GNNs) [Scarselli 08] which can process graphs of arbitrary shape and size, naturally bypass the issues and challenges encountered with MLPs for agent-agnostic control when the state and action space dimensionalities are not the same across robots.

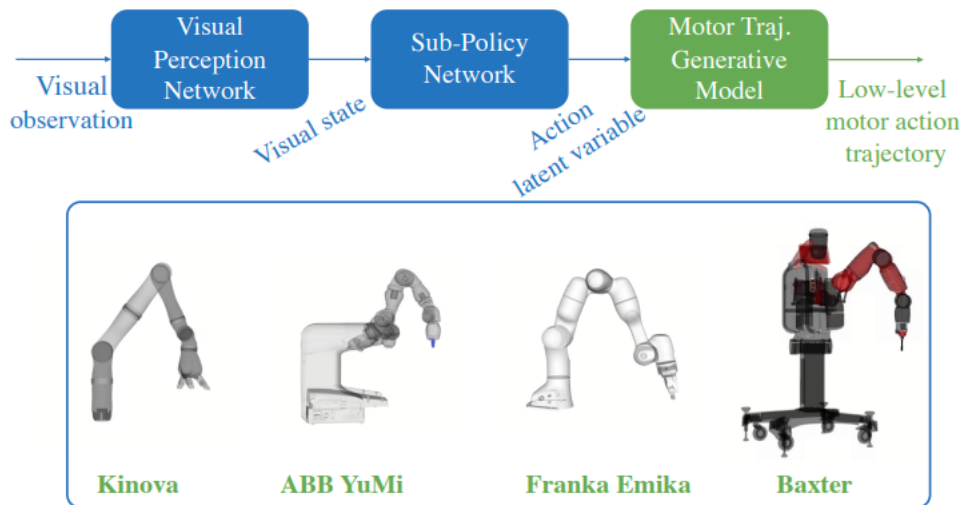


Figure 4.24: Top: schematic representation of the modular robot-agnostic policy. Bottom: robots considered for the experiments. Image taken from [Ghadirzadeh 21]

Notably, authors of [Wang 18] tested the zero-shot generalization capabilities of GNNs across multiple body configurations when learning locomotion skills. Each jointsbody has its own internal state derived from node-specific observations and the propagating state of neighbor nodes. At the end of the propagation phase, each actuator node computes a joint-specific action using its own policy. They experimented with centipede-like robots with varying number of legs, and demonstrated improved performance after direct transfer of the policy from one morphology to another.

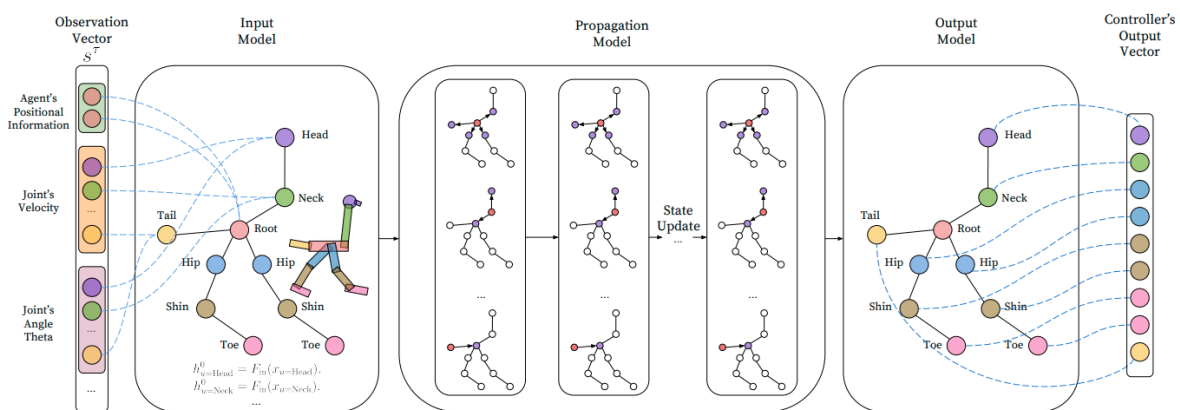


Figure 4.25: Example of graph structure for a Walker-Ostrich. Each node receives a set of specific observations, derives an internal state and propagates to its neighbors. The outputs from each controller are then concatenated to derive the full-body policy. Image taken from [Wang 18]

Similar work [Huang 20] decided to express the control policies as a modular neural network where each of its component is an identical reusable module instantiated at each of the agent's actuator. In this particular policy architecture dubbed Shared Modular

Policies (SMP), each module instance generates a local motor command from the actuator’s local sensors and coordination between modules is achieved using a learned message passing procedure. By training such a decentralized multi-agent population on a wide range of robot morphologies, they show that it is able to learn a shared control policy and generalize to unseen kinematic structure. In both previous work, the structure of the graph is based on the agent’s physical morphology. However, successor work [Kurin 21] studied the role of morphology in graph-based control and concluded that a morphology-based graph is not necessary and may even hurt performance. They argue that efficient messaging passing communication can be hard to learn in practice with such graphs. Instead, they propose to use an architecture based on Transformers [Vaswani 17], a special kind of GNNs [Battaglia 18], significantly outperforming previous works. [Trabucco 22] managed to improve on their results by framing morphology learning as a sequence modeling problem with tokenized actuators. The body representation is optimized using a RL-objective and the policy is modeled with a Transformer architecture. They showed that their method generalizes better to unseen morphologies when compared to prior methods, even exhibiting interesting zero-shot performance.

All the previously mentioned methods require training until convergence a policy on multiple possible robots of the training set, before a shared representation can emerge and be used for transfer on an unseen robot/task. In practice, it is therefore not clear if their method exhibits an actual learning speed up over a simple training from scratch on the target robot or task, as the amount of training needed prior to transfer could be significant. Furthermore, designing variations of the robots in the training set to help the policy generalize is not a trivial task as explained in [Jackson 21]. It requires expert knowledge to determine which physical parameters should vary and careful crafting of their distributions. Poorly handled parameters randomizations could even be detrimental for learning. Thus, it introduces new hyper-parameters to tune in the learning setting. In contrast, our method leverages the creation of a robot-agnostic latent space, ensuring instant generalization to new morphologies in a simple, fast and unsupervised manner by encoding time-aligned trajectories of a primitive task execution for the robots considered. There is no need for any physical parameters randomization and we only need to train the task module once, on one robot. Moreover, for most of the aforementioned methods, zero-shot results are reported for related but unseen morphologies (i.e morphologies slightly modified from a type seen during training). In contrary, the UNN module is trained on one robot and instantly generalizes to completely unseen robots.

4.5.2 Learning a correspondence mapping

Here we review CATL methods which focus on learning a mapping between states and/or actions of a policy in the source domain to obtain the policy in the target domain.

Work by [Zhang 21] proposed a novel method to learn cross-domain correspondence by using dynamic cycle-consistency, a form of adversarial training they have introduced. With unpaired and unaligned trajectories from source and target domains, they are able to learn correspondence across domains differing in representation (vision vs. internal state), physics parameters (mass and friction), and morphology (number of limbs) (see Figure 4.26). The learnt mapping is then used to adapt an expert source policy to the target domain. Experiments for cross-morphology transfers were performed on half-cheetah and swimmer, two agents from the MuJoCo library [Todorov 12].

One of the most recent papers in the field [Shankar 21], is concerned with a similar

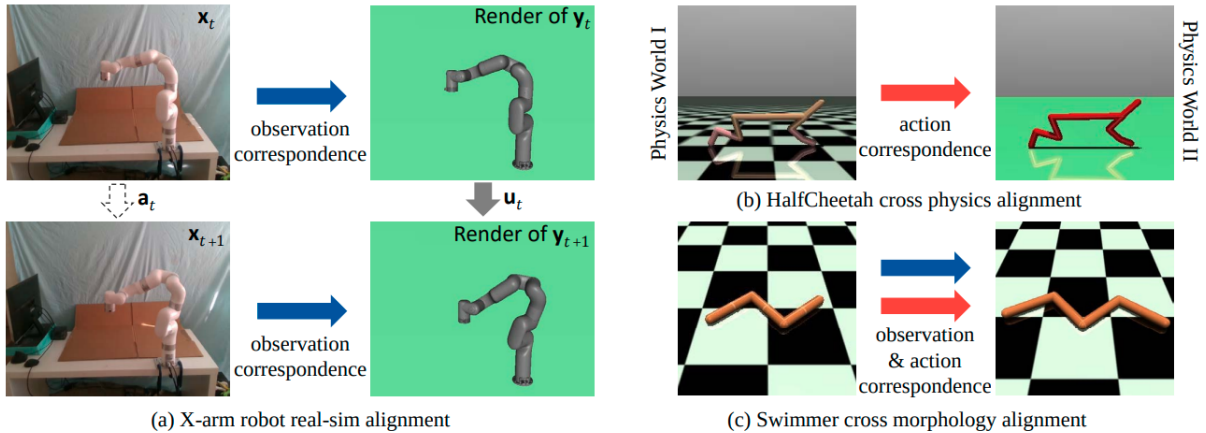


Figure 4.26: Experiments performed in [Zhang 21]: (a) sim2real visual adaptation, (b) cross-physic transfer and (c) cross-morphology transfer.

problematic as previous work: how to learn correspondences between morphologically different robots in an entirely unsupervised manner? However, rather than learning correspondences between states and actions, they focus on correspondences between skills. Inspired by novel advances in unsupervised machine translation (typically used in language processing), they build a translation model which helps transfer task-strategies across robots morphologies by translating skills from source to target domains. Using a more classical approach, authors of [Helwa 17] studied the properties of an optimal transfer learning mapping between source and target domains through a dynamical perspective. They theoretically analyze the properties and conditions of existence of such a mapping, and derive an algorithm to learn it from data and system identification. However, their method is limited to single-input, single-output systems and has not been shown to generalize to high-dimensional manipulators, only quadrotor platforms. Other works investigated the use of a lower-dimensional manifold to embed robot states and transfer knowledge. Specifically, [Bócsi 13] used Principal Component Analysis to obtain a source and a target manifold for the two robots considered. Then, a linear mapping is learned between manifolds to transfer task execution from source robot to target robot. A non-linear variant with Local Procrustes Analysis is explored in [Makondo 15] and [Makondo 18].

All aforementioned approaches use a correspondence mapping to transfer knowledge between domains. We argue that it is not a very scalable approach as it requires learning a mapping for each transfer direction (e.g one for UR10 \rightarrow Panda and one for Panda \rightarrow UR10). Moreover, some methods such as [Zhang 21] also require learning a forward dynamic model (which can be prone to compound errors) as well as action-specific and state-specific correspondence mappings. As a consequence, considering an additional robot for transfer entails learning an increasing number of mappings if we want every transfer direction to be possible (at least $2n$ mappings, with n the number of considered robots). On the other hand, with a shared latent space, adding a robot requires a constant number of mappings (2 in our case). As an analogy, we can think of a network of computers that need to communicate with each others. Using a shared latent space is the same as using a hub, while learning a mapping across domains requires a connection link between each possible pair of computers in a peer to peer fashion.

4.5.3 Using a common feature space

Our proposed robot-agnostic latent space is similar to the invariant feature space introduced in [Gupta 17]. However, their work is focused on transferring knowledge between robots through reward shaping by adding an "alignment" term to guide the student agent during training. As a consequence, their method does not enable zero-shot transfer unlike ours. Furthermore, they used simple autoencoders which are prone to overfitting and less expressive due to their lack of regularization. In contrast, we make use of VAE for all the reasons detailed in section 4.2.2. As a follow up to their work, [Hu 19] proposed to use paired variational encoder-decoder models that disentangles the control of robots into shared and agent-specific latent spaces. By doing so, they argued that they remove the individual factors from the shared latent space to improve knowledge transfer through reward shaping. This is similar in spirit to work by [Bousmalis 16] on domain adaptation, except that they did not explicitly enforce orthogonality between shared and individual latent space. Inspired by several prior works on policy distillation [Liu 19], [Rusu 15] [Wan 20] proposed to augment representations in the layers of the student network with useful representations from the layers of a teacher network by using lateral connections between their respective policy and value networks. As such, knowledge can "flow" between a teacher pre-trained on a related task and a single student agent. Their main contribution to previous works is the use of an embedding-space such that $|S_{emb}| = |S_{teacher}|$ to handle the mismatch when the state-space dimensions of the teacher and student are different. First, the encoder is trained to produce task-aligned embeddings by utilizing the policy gradient to update embedding parameters. Then, to maximize correlation between the embedding vectors and their input states, a mutual information objective is maximized. The authors argue such a high-correlation helps for knowledge transfer. During student training, transfer is performed by first deriving an embedding vector from an observed student state and then feeding it to the teacher network to extract the corresponding hidden representation. Extracted pre-activation outputs are then mixed to the student hidden representation to guide its learning process. They tested their approach on six morphologies on a locomotion task (see Figure 4.27). While significantly improving sample-efficiency, their method does not exhibit zero-shot performance contrary to ours.

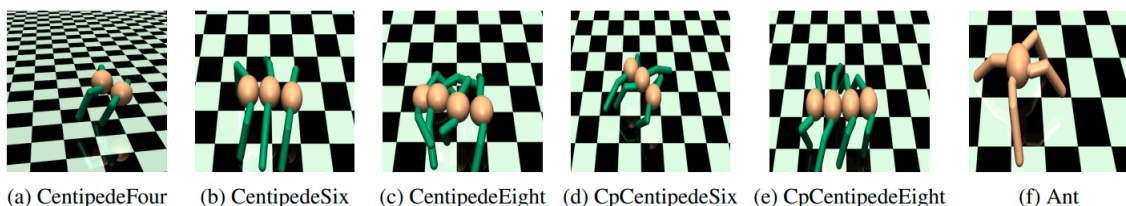


Figure 4.27: Morphologies considered for the locomotion task in [Wan 20]

Building on top of a broad range of topics, including transfer learning, imitation learning, and information theoretic RL, [Hejna 20] proposed a method to hierarchically decouple high-level transferable knowledge from low-level robot-specific knowledge. Their method is conceptually very similar to ours and [Mounsif 19a], but fundamental differences exist:

- They consider a goal space (i.e a robot-agnostic space) which needs to be defined by an expert (effector position/velocity, torso position etc..). We opted for a learned robot-agnostic space to avoid the need for expert knowledge.

- They minimize the mutual information between the morphology and the behavior to enforce decoupling between high-level and low-level modules and encourage robot-agnosticity of the high-level policy. In our case, robot-agnosticity of the high-level task module is ensured by training it into a shared latent space.
- They tackle catastrophic forgetting during fine tuning of the task module on the target robot, by adding a KL-constraint such that the fine-tuned policy stays close to the original one. We addressed the same issue by using a trust-region based RL algorithm such as PPO, which optimizes a policy under a KL-constraint.

4.6 Conclusion

This chapter discussed some of the limitations of the UNN framework as proposed by Mehdi Mounsif during his PhD. While his approach is very effective and has proven to be successful for transfer learning between robotic arms, it relies on hand-crafted features to define the agent-agnostic feature space. Motivated and inspired by several prior works, we introduced a simple and yet efficient methodology to overcome this shortcoming and transfer knowledge between agents regardless of their body morphology. At the core of our approach: a robot-agnostic latent space constructed from time-aligned demonstrations on the considered robots, followed by the training of the task-module within it. Regarding sample-efficiency, our method outperformed by a large margin the baseline models composed of state-of-the-art RL algorithms. In particular, our experimental results showcased (i) close to baselines zero-shot performance in most of the transfers considered, (ii) significantly less required training samples than baselines, when fine-tuning is needed to recover performance and (iii) slight sample-efficiency increase when training an agent from scratch thanks to the prior knowledge included in the bases. Overall, our approach demonstrates an undeniable superiority in terms of sample-efficiency over regular RL training and does not require a large additional computation cost to setup.

However, several challenges remain to be solved. Despite very convincing results, directions for future improvements are obvious when looking at the data. Our approach suffers from: (i) occasional lower asymptotic cumulative rewards which indicates a less optimal policy with respect to the RL objective and (ii) over-fitting to the source domains, which requires a thorough testing of the model’s checkpoints on the targeted robots. From a practical point of view, using unpaired and/or unaligned trajectories (instead of time-aligned demonstrations) for latent space building may prove a valuable addition to our framework. Some preliminary results presented in Appendix A are available when dealing with unaligned trajectories. Addressing all these issues will motivate future works. Additionally, it could also be interesting to investigate more sophisticated types of VAE models to study the impact of the structure of the latent space for downstream model optimization and transfer. For instance, [Davidson 18] proposes to use a hyperspherical latent space using a von Mises-Fisher prior distribution to better represent directional data such as joint configurations or velocities. They argue that a more uninformative prior could better capture the true data manifold unlike a biased Gaussian prior.

Chapter 5

Delay Aware Universal Notice Network

In the previous chapters, we introduced the UNN framework along with the LS-UNN extension as effective methods for dealing with CATL. We have shown that zero-shot generalization was possible for sim2sim transfers. Additionally, despite the reality gap, sim2real transfers also exhibited interesting performance. However, the manipulation task considered with the physical robots did not pose much issues from a sim2real point of view and was straightforward to implement. Many applications involving transition from simulation to reality are significantly more challenging. One major issue is the delay on the physical robot that may deteriorate the performance of the deployed agent. In this chapter, we introduce the Delay Aware Universal Notice Network (DA-UNN), which deals with delays immanent to physical systems in order to improve sim2real transfer. We evaluate the efficiency of our approach using simulated and actual robots on a dynamic manipulation task where delay management is crucial. As this delay-management approach was developed before the LS-UNN, all experiments only feature the regular Cartesian UNN method.

5.1 Motivations

General purpose simulators provide cheap training data to learn complex robotic skills. However, simulation to real world transfer is still an open problem. In general, simulations are imperfect and difficult to calibrate. The resulting modeling discrepancies cause a reality gap, which makes the transfer of RL policies from simulation to the real-world non-trivial. Most of the time, sim2real methods focus solely on domain adaptation between the real world and the simulation [Bousmalis 18, Arndt 20]. They tend to ignore troublesome hardware specific issues such as control latency induced by medium of data transmission, computation delay, sensor sampling rates (etc.) unmodeled in the simulator. Consequently, the policy obtained by training in simulation could be drastically disturbed once transferred in the real world if the task requires short reaction time.

In this contribution, we considered the time delay associated with the physical system as another model’s input by including it in the observed state. At training time, we randomize the value of the delay and show that the agent is able to adapt to multiple delays on a dynamic and delay-sensitive manipulation task. Our contributions are as follows:

1. We present and evaluate a delay-aware method to deal with the immanent delay on real hardware, thus furthering the adaptation capabilities of the UNN.

2. We evaluate the benefits of the UNN multi-robot transfer method over a vanilla transfer on real world robots. A pool containing four differently shaped real and virtual robots is tested against a dynamic manipulation task they have not been trained on, by using the knowledge created by another agent as depicted in Figure 5.3.

5.2 DA-UNN

5.2.1 Constant Delayed Markov Decision Process (CDMP)

The standard UNN and LS-UNN proved their efficiency and versatility on a broad panel of tasks in simulation and on physical robots. However, these results were obtained with perfect robots (e.g no offset and no delay) acting in a standard Markov Decision Process (MDP). Traditionally it is assumed in RL that at every timestep, the environment pauses while the agent receives the current observation, in order to derive an action that will be executed without delay. Naturally, things do not behave this way in the real world. All agent observations and actions are delayed by an amount depending on the hardware used for the task and for perception. Therefore, an agent trained in simulation without exposition to delays will perform worse or even fail in the real world if the task requires fast reaction time. This brings up the need to adopt a different decision process modeling to solve tasks in the presence of delay. As we consider the delay to be constant, we found the Constant Delay MDP formulation introduced in [Walsh 09] to be well suited. A CDMPD enlarges the regular MDP formulation described in 2.5.2 with an extra parameter $d \in \mathbb{N}$ defined as the number of timesteps between an agent occupying a state and receiving its feedback from the environment. As such, d models the delay governing the environment and a CDMPD is defined by a 6-tuple $CDMPD = \langle S, A, P, R, \rho_0, d \rangle$. Naturally, when $d = 0$ we recover a regular MDP. A known result in CDMPD is that observation delay and action delay are equivalent from the agent's point of view [Katsikopoulos 03]. Hence, we treated the total delay as being entirely caused by observation delay (see Figure 5.1).

5.2.2 Solving a CDMPD

Introducing the notion of delay inside a Markov decision Process raises new challenges. When observations are delayed, the agent must predict an action only based on outdated information about the environment. Consequently, a CDMPD can be seen as a Partially Observable MDP, as the agent does not act with a complete knowledge of the environment's state. According to [Katsikopoulos 03], we can transform a CDMPD into an "augmented" MDP by expanding the state space such that it recovers the Markov property. This approach relies on the so-called information-state $I_d \in S \times A^d$, where the state space S is augmented with the history of the d last actions taken since the last delayed observation. More formally, given a MDP with constant delay $C = \langle S, A, P, R, \rho_0, d \rangle$, we can construct an equivalent undelayed MDP $M = \langle I_d, A, P, R, \rho_0 \rangle$ with $I_d = (s, a_1, \dots, a_d)$ for $s \in S$ and $a_i \in A$. It is theoretically possible to derive an optimal policy $\pi(\cdot|I_d)$ for the CDMPD [Bertsekas 00] by acting conditioned on the information-state I_d , rather than the original state-space S .

5.2.3 Delay Aware UNN

While this approach is sound and theoretically motivated, it does not allow direct transfer between systems with different delays, as the input dimension depends on d . Moreover, it induces an exponential growth of the state-space when d increases. As a result, exploration of the state space is much more challenging, as the agent may need to experience state-actions in $S \times A^d \times A$, requiring more samples to converge. In this work, we address the delay issue by augmenting the state space of the UNN module with the estimated delay of the system and by training the agent on a corresponding delayed environment as depicted in Figure 5.1. A key feature of the UNN is its ability to adapt to any robot regardless of its morphology. To keep this idea of "universality", the delay was randomized during training to ensure that the UNN can adapt to a wide range of delay. By giving it access to the immanent delay, we enable the UNN to act accordingly and to develop predictive capabilities. Thus, we add d to the task specific observations.

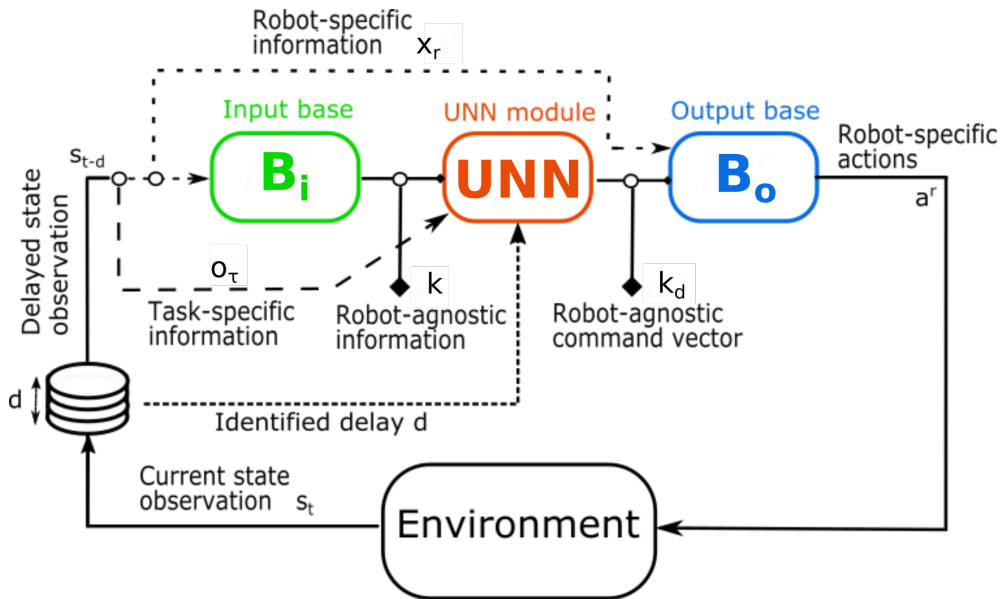


Figure 5.1: Schematic representation of the delay aware UNN. Observations are queued into a pile of length d and at each timestep the observation at the top is fed to the agent (first in, first out).

During training, the delay is sampled regularly from a discrete uniform distribution $\mathcal{U}(d_{min}, d_{max})$ where d_{min} and d_{max} are respectively the minimum and the maximum delay considered for the environments. Since there is no assumption about the systems, we assumed a uniform distribution of the delay. But any prior knowledge could be used to deduce a more suited delay distribution. When deployed, the identified delay of the system is fed to the UNN, so that it is "aware" of the delay it is working with and can act accordingly. While our approach can only yield sub-optimal CDMDP policies due to the hidden information and incomplete state space considered, we believe that it represents an interesting trade-off between optimality and flexibility. This very simple method can improve drastically the performance of an agent on a delayed MDP as presented in section 5.4, given that the delay has been accurately determined. Moreover, it can generalize to a wide range of delays and as such, is suited for transfer on systems with different hardware and on different robots.

5.3 Experimental setup

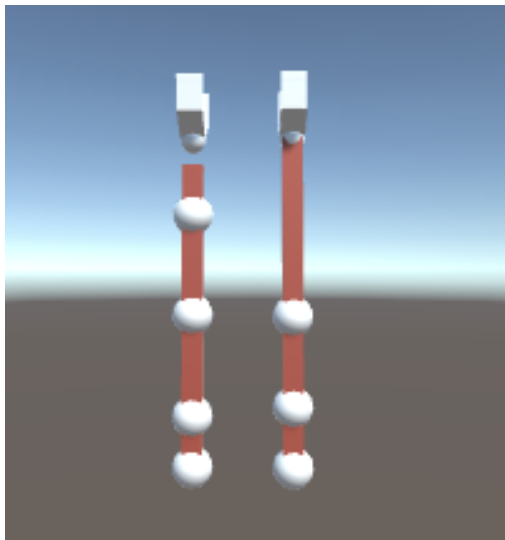
5.3.1 System Architecture and Robots

In this section, we briefly present the different robots adopted throughout these experiments. We tested our method on both physical and simulated robots to demonstrate its efficiency and versatility. We consider a serial arm braccio robot with 5 DoF and a 4 DoF serial arm (see Figure 5.2b). These DIY robots are cheap and usually hard to work with, given their low reliability. Still, we manage to use them efficiently in our experiments. We also consider their simulated counterparts (see Figure 5.2a and 5.2b).

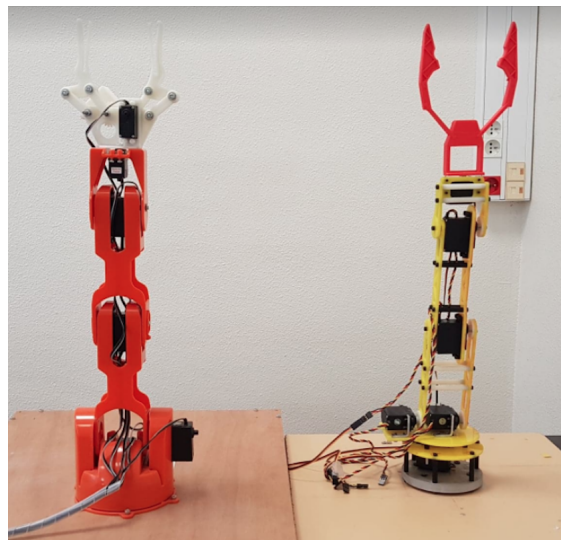
We also consider the virtual BAM robot on top of the 4 other robots:

- **BAM**: the virtual BAM robot with the identity bases.
- **Robot 1**: the virtual braccio robot.
- **Robot 2**: the virtual 4 DoF robot.
- **Robot 3**: the physical braccio robot.
- **Robot 4**: the physical 4 DoF robot.

Regarding the cheap robots, significant offset was present in the robots joints, making each movement inaccurate. In this regard, the offsets first needed to be identified, in order to use analytical models efficiently on both physical robots.



(a) Robot 1 (left) and 2 (right).



(b) Robot 3 (left) and 4 (right)

Figure 5.2: Robots considered for the experiments

A fixed webcam was used to obtain the required pose estimations with OpenCV. The control frequency was 10 Hz, which means the agent was observing the environment state and acting every 0.1 second. The nominal delay was in average 300 ms on the physical systems. We identified the delay by measuring the time between a command sent to the robot and the observation by the agent that the robot moved.

On the simulation side, agent’s training was performed in the Unity physic engine with the ML-agent package. The PPO algorithm [Schulman 17] was used to create the

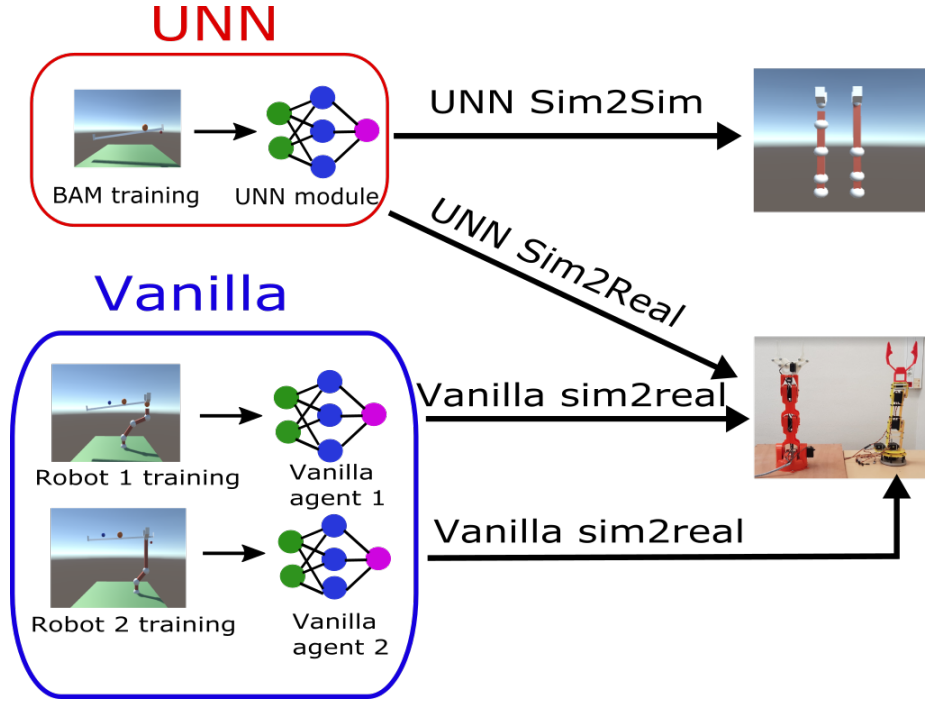


Figure 5.3: Transfers considered for the robots. UNN module is trained with the BAM robot and then transferred to all robots. Vanilla agents are trained directly on the simulated robots and then transferred on the corresponding physical robots.

neural network policies, as it provides a monotonous performance improvement while being perfectly adapted to continuous action spaces. We trained four kinds of agents:

- **Delay Aware UNN Agents:** The BAM virtual robot is trained in simulation with exposition to randomized delays to create the UNN.
- **Delay Aware Vanilla Agent:** The agents are trained from scratch directly on the simulated robot, with exposition to randomized delays.
- Finally, we also considered their **delay unaware** counterparts, trained without exposition to delays, in order to display the benefits of our delay management approach.

These agents will be used for the transfers detailed in section 5.4.2 (see Figure 5.3).

5.3.2 Task description

We display our method benefits on a 2D manipulation task (planar task), where a robot needs to keep a ball at a desired position on a gutter. In this regard, only 3 DoF are required for the physical robots (base rotation and wrist roll are unused). To further increase the gap between both robots, the Robot 4 is used as a 2 DoF robot (wrist pitch is unused). The gutter is fixed at one end and held at the other end by the robot’s effector which therefore decides of its orientation and, as a consequence, of the position of the ball (see Figure 5.4). This task can be formalized with the following MDP:

State: $s_t \in \mathbb{R}^{4+1}$: the ball position and velocity on the gutter, the effector height, the desired ball position and the system delay d for the delay aware agents.

Action: $a_t \in \mathbb{R}^n$ is the target joints position (n being the number of considered joints). The vanilla agent was not making any progress with a full access to the action space. Indeed, to balance the ball on the gutter, it is first needed to hold it properly. These desired body configurations are just a fraction of the full state space and it is very unlikely to discover them without any prior knowledge of the task. To ease the vanilla agent learning process, its action space has been constrained to output joints offset values w.r.t a reference joints position which maintains the gutter in an equilibrium position.

Reward:

$$r_t = \begin{cases} r - \beta|\theta_e| & \text{if } d_{des,b} < \delta \\ -\alpha d_{des,b} - \beta|\theta_e| & \text{else} \end{cases} \quad (5.1)$$

where r is a small positive reward, δ is the positive reward area and $d_{des,b}$ is the distance between the ball and the desired ball position. $|\theta_e|$ is the angle between the effector pose and the vertical plane and α and β are weighting constants. This penalty ensures that the effector is in the right orientation to hold the gutter properly for the vanilla agent. In contrast, the effector orientation constraint for the UNN is handled by the output base, which means that β is set to zero when training the UNN.

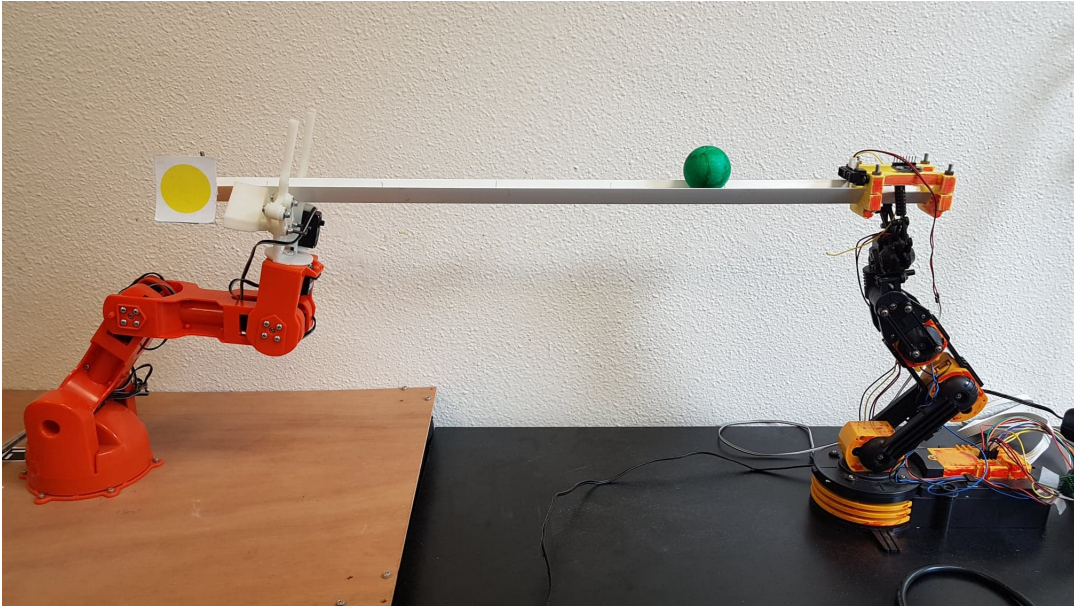


Figure 5.4: Physical experiments setup. Left robot performs the task, while the right robot is used only to hold one end of the gutter.

5.3.3 Delay Aware UNN creation

As the BAM method showed better transfer results [Mounsif 19a], we decided to use it to create the UNN module. As a recall, in Base Abstracted Modeling, the robot is assimilated to its effector. For this task, we chose the intermediate states $k \in \mathbb{R}$ and $k_d \in \mathbb{R}$ shared between the UNN module and the bases to be a single value indicating at what height below or above the horizontal reference position of the gutter the effector is/should be. The UNN module was receiving an extra input $d \sim \mathcal{U}(0.1, 1)$ representing the current

delay of the system during training. A delay range between 0 and 1 is recommended as it corresponds to a normalized input. In our case, it also corresponds to our actual delay in second, with 0.1s being the smallest delay possible given our control frequency. The delay was created in the simulator by stacking the observations in a FIFO buffer, before feeding them to the UNN module.

5.4 Results

In this section, we present our results both on training and transferring on the chosen manipulation task. In particular, we compare the UNN agents with the vanilla agents with and without delay awareness. For further experiments, delay was added artificially to the real system with the same FIFO method seen in section 5.3.3. Code can be found at github.com/sabeaussan/DelayAwareUNN. Videos showing our results are available here.

5.4.1 Training

During the training, the desired ball position and system delay (for delay aware agents) were regularly changed to improve the adaptive capabilities and re-usability of the UNN. More precisely, a new delay d was sampled from $\mathcal{U}(0.1, 1)$ every 15 episodes. The desired ball position given to the model, varying between 20% and 80% of the gutter length, was also sampled from a uniform distribution $\mathcal{U}(0.2, 0.8)$ every 1000 training steps. Both the BAM agents and the Vanilla agents were trained for 4 millions steps. Figure 5.5 shows the cumulative reward obtained per episode. Only the term $d_{des,b}$ (distance between the ball and the desired position) common to both reward functions was considered for the comparison, as it reflects the agent overall progression on the task. As shown in Figure 5.5, the BAM agents in both settings converge slightly faster than their vanilla counterparts. The BAM agents focus solely on the task, leaving robot specific considerations to their bases. This decomposition of the learning problem similar to hierarchical RL eases the learning process. It is also worth noting that introducing varying delay during training reduces the convergence speed, as the task becomes more challenging. However, in the UNN framework, this training overhead is outweighed by the increased reusability of the UNN module.

5.4.2 Transfer

There are two kinds of transfer to consider: simulation to real robot transfer and robot to robot transfer. The UNN framework mitigates the sim2real transfer problem by considering the real robot and the simulated one as two different robots, each one with its own bases, thus partially addressing the sim2real transfer as a robot to robot transfer. In this section we evaluate two methods of transfer

- **UNN transfer:** Once trained until convergence with the BAM robot, the UNN module is transferred to each robot of the set.
- **Vanilla transfer:** The vanilla agents trained on the simulated robot are directly transferred to their physical counterpart. This will serve as a baseline to study the UNN benefits for sim2real transfer.

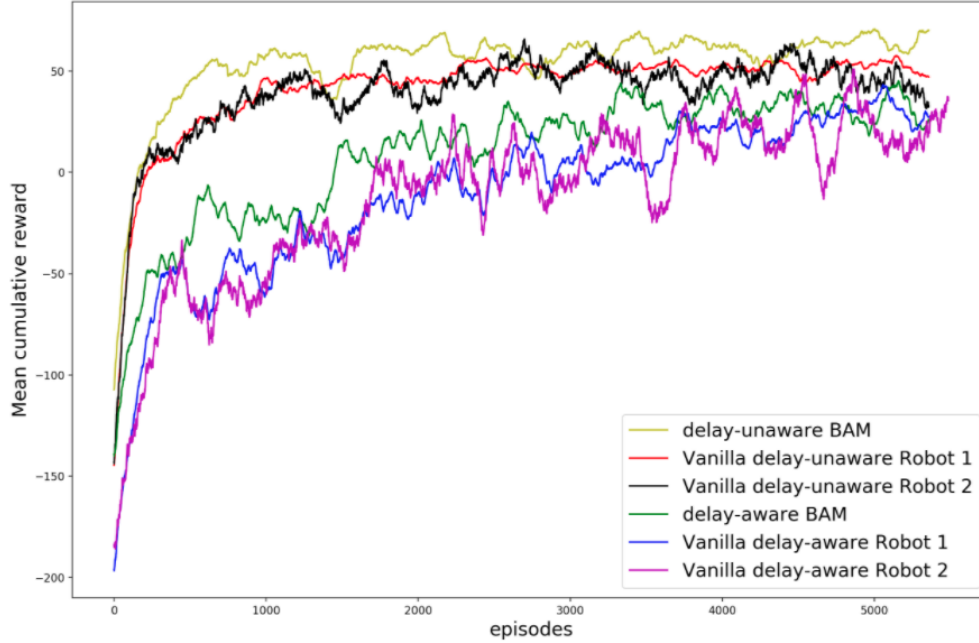


Figure 5.5: Training curves. All the agents were trained for 400000 steps.

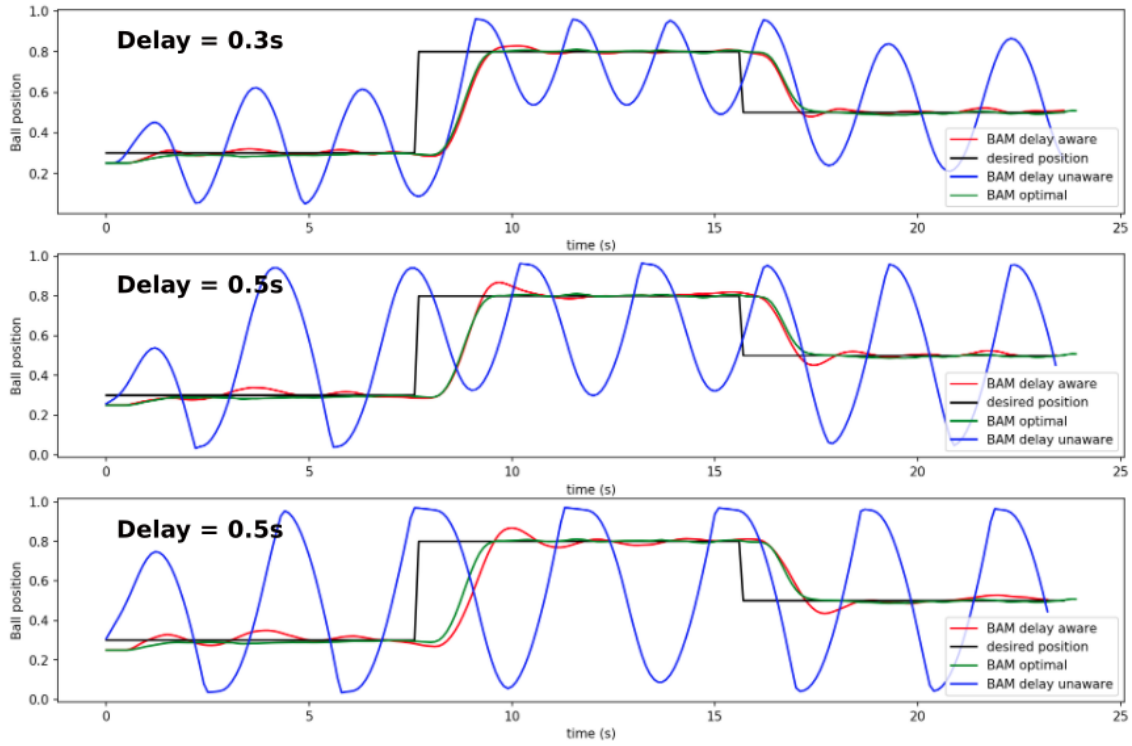
The performance metric used is the integral of the absolute value of the error between the ball position and the desired ball position over time. This metric has the advantage of taking into account both settling time and the steady state error (the closer to 0, the better). For a fair comparison, each experiment has been conducted with the same settings (same initial ball position and desired ball position). Performances displayed in Tables 5.1 and 5.2 are averaged over 50 episodes.

Influence of delay:

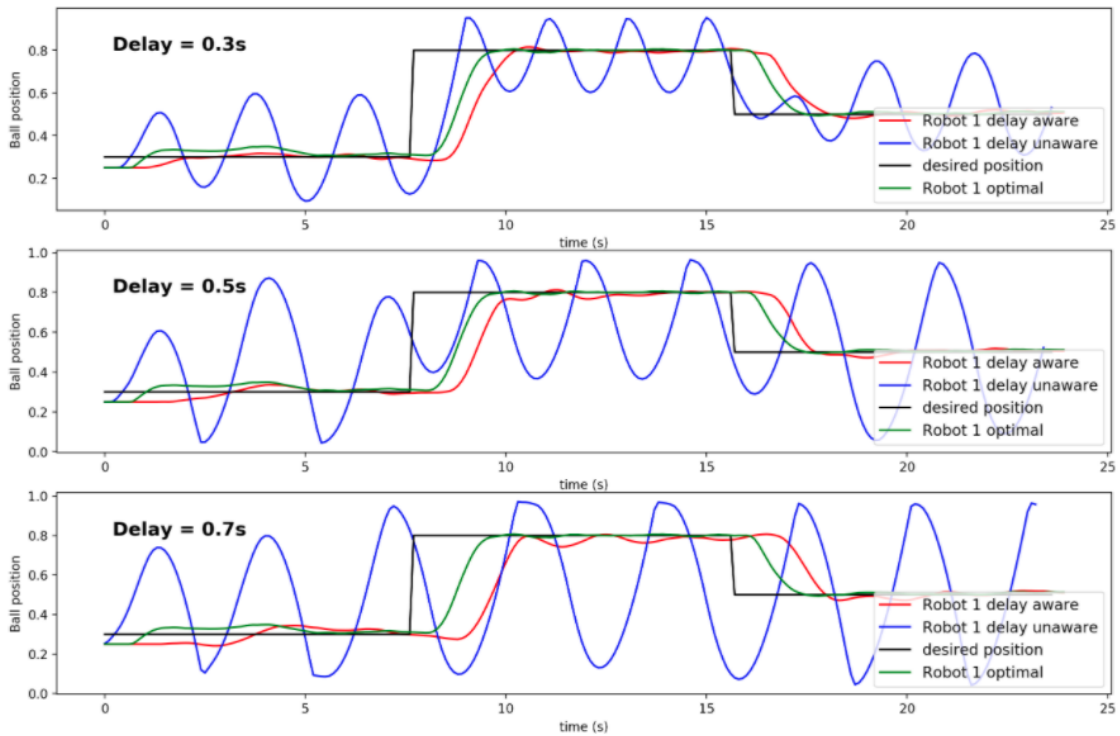
Robots/Delays	0.3	0.5	0.7
BAM	3.12 / 15.57	3.94 / 22.57	4.91 / 25.28
UNN Robot 1	3.26 / 11.84	3.96 / 20.12	4.98 / 23.48
UNN Robot 2	3.43 / 12.21	4.19 / 18.47	5.10 / 21.19

Table 5.1: Sim2sim transfer. Performances obtained for the UNN transfer on the simulated robots. Results are displayed with delay aware method on the left / delay unaware method on the right.

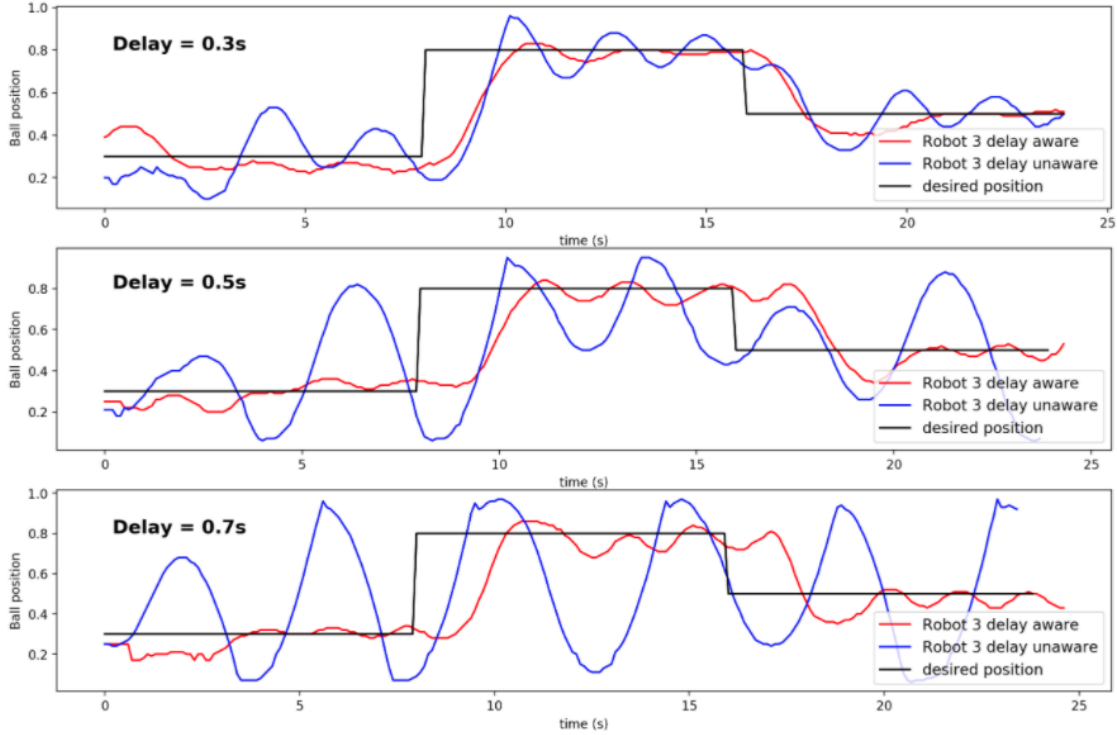
In this part, we evaluate the contribution of the DA-UNN for delay management, on both simulated and physical robots. Three delays were considered for the experiments: 300 ms (corresponding to the delay on the physical system), 500 ms and 700 ms. Figure 5.6 shows ball trajectories for the three delays considered, obtained by the UNN agents on robot 1 (virtual braccio robot) and 3 (physical braccio robot). On the simulation side, we added an optimal trajectory obtained with the delay unaware UNN agent acting on an undelayed environment to serve as a reference (see Figure 5.6b). The same agent was then exposed to the delays considered to study how performance deteriorates for



(a) BAM robot



(b) UNN Robot 1



(c) UNN Robot 3

Figure 5.6: Ball trajectories with 0.3, 0.8 and 0.5 as desired ball positions.

unaware agents as the delay increases. As shown, agents not exposed to delays during training completely failed and systematically overshoot when trying to get the ball at the required position in delayed environment. Figure 5.6c emphasizes the inability of the delay unaware agents to cope with the physical system immanent delay (300 ms) as the ball starts oscillating. Moreover, as the delay increases, the delay unaware agents tend to become unstable. As for delay aware agents, in the simulation, they still manage to follow closely the optimal trajectory.

Table 5.1 shows the performances obtained in sim2sim transfer with the UNN agents on both delay aware and unaware settings. It is shown that delay aware agents perform from 3.5 to 5.72 times better than their unaware counterparts. It is also clear from looking at Figure 5.6c and Table 5.2, which shows the average performance after sim2real transfer, that dealing with delay in simulation greatly improves the results of the UNN agents once deployed on the physical robots. Vanilla agents also benefited from this delay management method, as shown in Table 5.2b, demonstrating the versatility of the proposed method.

sim2sim transfer

In this paragraph, we discuss the results obtained when transferring the delay aware UNN module from the BAM robot to robots 1 and 2 in simulation. We also compare the performance obtained against delay aware vanilla agents which learned the task from scratch on robots 1 and 2. As shown in Tables 5.1 and 5.2b, UNN-based approaches slightly outperform the policies of the vanilla agents for the robots and delays considered. We want to emphasize that the UNN module has been trained only once and on only one robot, the BAM robot, but still performs better than the vanilla agents specifically trained on robots

Robots/Delays	0.3	0.5	0.7
BAM	3.12 / 15.57	3.94 / 22.57	4.91 / 25.28
UNN Robot 3	4.78 / 9.88	5.05 / 22.32	8.02 / 24.43
UNN Robot 4	5.86 / 15.72	7.45 / 22.42	8.76 / 24.41

(a) UNN transfer: BAM \rightarrow robot 3 and BAM \rightarrow robot 4

Robots/Delays	0.3	0.5	0.7
Vanilla Robot 1	3.32 / 17.33	4.22 / 26.17	5.56 / 31.48
Vanilla Robot 3	5.43 / 19.65	5.97 / 28.22	9.33 / 33.43
Vanilla Robot 2	3.78 / 18.63	4.81 / 26.45	6.16 / 32.48
Vanilla Robot 4	7.58 / 21.13	9.55 / 27.05	10.62 / 33.82

(b) Vanilla transfer: robot 1 \rightarrow robot 3 and robot 2 \rightarrow robot 4

Table 5.2: Sim2real transfer. Performances obtained for the vanilla transfer and the UNN transfer on the physical robots. Results are displayed as delay aware method on the left / delay unaware method on the right.

1 and 2. These results demonstrate the appealing re-usability and effectiveness of the UNN module. In some cases, the delay aware UNN agents achieve zero-shot performances (e.g robot 1 with delay 0.5). In the worst case, the transfer efficiency is 90.9% (3.12/3.43), 100% being the performance obtained by the UNN module on the BAM robot. In average, the transfer efficiency is 97.7 % for robot 1 and 93.7% on robot 2. Ideally, the UNN module paired with any of the robots would yield similar performance as with the BAM robot. However, in some cases the body configurations required to comply with the UNN commands are not precisely achievable by the robot. For instance, the desired effector position may need some of the joints to rotate beyond their limits. This also explains why the UNN transfer is less efficient on the 2 DoF robot, as it is less expressive and has a harder time following UNN commands.

sim2real transfer

In this paragraph, we study the UNN methodology as a sim2real transfer tool. More specifically, we compare the performance obtained after transfer on the physical robots for Vanilla agents and UNN agents. In this case, both the UNN module obtained on the BAM robot and the vanilla agents obtained on the simulated robots, were transferred to the physical robots. As shown in Figure 5.6, the UNN agent on robot 3 still manages to put the ball at the desired positions without too much overshooting. Table 5.2 shows the results obtained. As usual, the agents trained in simulation and transferred to the real world show lower performance than their virtual counterparts due to the reality gap. However, they still manage to obtain decent performances. One notable result is that the delay unaware agents transferred to the physical robot obtain very poor performance unlike their delay-aware counterparts. Once again, UNN based agents outperform vanilla agents. For instance, the UNN based transfer reaches up to 78% (100% corresponds to the BAM performance) in the best case, while the vanilla transfer reaches 70.6% (100% corresponds to the vanilla agent on robot 1). In average, the UNN sim2real transfer

efficiency is 68% on robot 3 and 54% on robot 4, against 63% on robot 3 and 52.7% for robot 4 for the vanilla sim2real transfer. As mentioned earlier, this slight sim2real improvement can be attributed to the robot-agnostic nature of the UNN module. Indeed, even if the vanilla agents were trained in simulation with a virtual copy, it remains an inaccurate model of the physical robot. The UNN module on the other hand ignores those discrepancies by considering the physical robot and the virtual one as two different robots, each with their own bases.

5.4.3 Discussion and perspectives

From the previous results, it clearly appears that the delay management method used considerably improves the performances when working with delayed environment, as it is often the case in the real world. Moreover, the UNN approach not only achieves very efficient transfer between robots in simulation, but slightly improves sim2real transfer over vanilla transfer. However, the zero-shot sim2real transfer efficiency is nowhere near what was obtained for the sim2sim transfers. Nevertheless, further training could be done on the physical robots to achieve better performance. As aforementioned, the UNN mitigates the sim2real transfer by considering the physical system as just another robot that can be interfaced with the UNN module. Nevertheless, the UNN module which was trained in simulation can still overfit on its environment. As a result, the instructions given can be unsuitable if it is placed in a new domain with a slightly different state distribution, e.g the real world. Fortunately, the UNN approach can be combined with state-of-the-art sim2real methods such as automatic domain randomization[Akkaya 19] to improve sim2real transfer.

5.5 Relation to prior works

In this section we review works that address the issue of delay management. Several approaches have been proposed in the context of Machine Learning. Roughly speaking, they can be classified in two categories: Model-based and state-augmented approaches.

5.5.1 Model-based approaches

This subsection presents an overview of methods implementing a model to predict future states given a history of past actions or states. By doing so they assume a regular MDP and develop a controller oblivious to delay. A cornerstone paper in RL with delay [Walsh 09] learns a model of the undelayed Markov Decision Process (MDP) to simulate the most likely state in which the agent currently is, given the last observed (delayed) state and the k last actions taken since, k being the delay in timestep. This allows the agent to take decisions based on the expected current state rather than an outdated state, effectively undoing the harmful effect of delays. They also introduced the concept of Constant Delayed MDP presented in Section 5.2.1. In [Behnke 04], the authors proposed a neural network-based method to address the control delay issue. Their contribution emerged from the practical need of controlling a robot during the RoboCup small size league. Indeed, the feedback control system composed of computer vision, motion control and communication, introduced a delay ranging from 100 ms to 150 ms. To address this issue, a predictor approximated by a neural network (more specifically a multi-layer perceptron), must infer the current state of the fast moving robot by observing a vector

of stacked outdated states and the last undelayed action. Given the predicted state and a simple PID controller, they were able to successfully cancel the adverse effects of delay and control the robot. In [Firoiu 18], the authors have voluntarily introduced action delays to evaluate the performance of an agent with human level reaction time. To highlight their contribution, they consider a highly-dynamic and fast paced multi-player fighting video game. Unlike previous work, they solve the delay problem by using a state-predictive model based on a multi-layer Gated Recurrent Unit [Cho 14]. The predicted state is then fed to a regular policy to compute the action a_{t+d} , with d the delay. Similarly, the authors of [Derman 21], recursively apply a one-step ahead forward model of the environment to predict the future state the agent will be in when the delayed action will be executed. The authors of [Chen 21] proposed a model-based approach to solve a state-augmented MDP (which they call Delay-Aware MDP) in the presence of action delay (see Figure 5.7). The transition function is decomposed into a known part which corresponds to the dynamics induced by the action delay, and an unknown part which corresponds to the dynamics of the delay-free MDP. By leveraging SOTA model-based methods they were able to learn the unknown dynamic and solved the delay MDP with minimal degradation of performance. Additionally, the learned dynamics model is transferable between systems with different delay steps. However, their method does not enable transfer between morphologically distinct entities. Furthermore, all of these previous methods rely on a predictive model to solve the delay issue. In addition to being hard to learn and subject to compounding errors, a predictive model is explicitly trained on a particular system, preventing any Cross-Agent Transfer Learning.

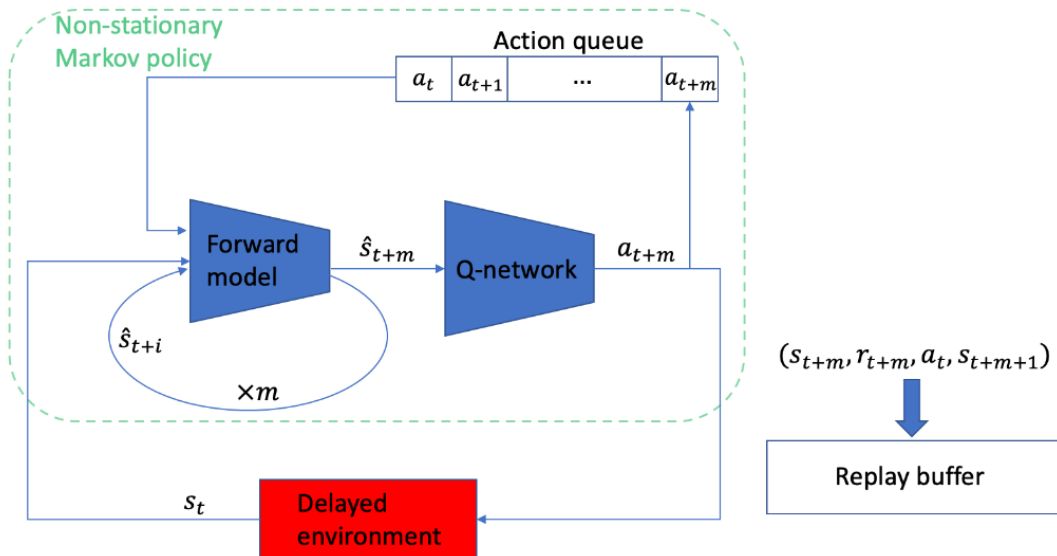


Figure 5.7: RL policy (Q-Network) using a forward model to act on undelayed observations [Derman 21].

5.5.2 State-augmented approaches

In this subsection, we review methods that directly rely on the information-state approach presented in section 5.2.2. As such, they leverage policies conditioned on a buffer of pending actions and the last observed state to solve the CDMDP. Ramstedt and Pal [Ramstedt 19] studied real-time RL by taking into account the computation time needed

to select an action (supposed inferior to one time step). Their proposed algorithm, Real-Time Actor-Critic, additionally takes as input the action from the previous step in order to compensate the one-step action delay. They demonstrate the efficiency of their proposed method on a real-time autonomous driving simulator. Successor work [Bouteiller 21] studies reinforcement learning in the presence of random delays. They argue that off-policy learning algorithms, even when using the information state $I_d = (s_0, a_1, \dots, a_d)$ may struggle to converge. They hypothesize that this is due to the credit assignment problem (i.e how to attribute an outcome to a delayed action). By partially re-sampling fragments of trajectories from the experience buffer, they can estimate the value function on-policy to fix this issue. Experiments clearly show the superiority of their approach against the vanilla state-augmentation approach. Related work [Nath 21], proposed the delay-resolved DQN, an extension to the regular deep Q-Network algorithm acting based on the information state I_k rather than the current state (see Figure 5.8). They demonstrate the effectiveness of their approach on gym environments [Brockman 16] over regular DQN for constant-delay MDP. Additionally, they claim that their algorithm is computationally cheaper than previous approaches. However, these previous methods are not suited for transfer between systems with different delays. The agent must learn again from scratch whenever the system delay changes as the size of I_d depends on it. In contrast, our proposed method is able to handle a range of different delays with randomization. This is similar in spirit to [Akkaya 19], where authors account for the imperfect actuators of the real hardware by introducing a fixed action delay with a probability of 0.5 at the beginning of every simulation training episode.

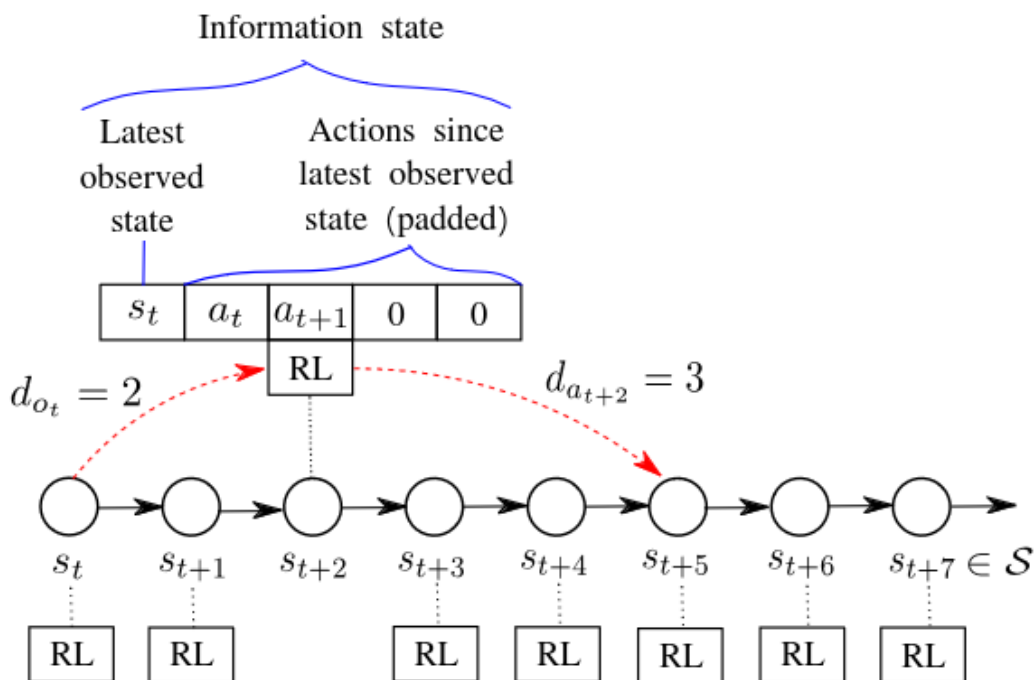


Figure 5.8: RL policy acting on a environment with observation delay using information-state I_2 to derive a delayed action [Nath 21].

5.6 Conclusion

In this chapter, we studied the benefits of the UNN transfer for a sim2real application. More specifically, we addressed the delay management problem that occurs when working with a physical system by making the UNN "aware" of the latency of the system it is working with. By doing so, we extended the versatility of the UNN method and the range of compatible systems. We demonstrated this method efficiency by solving a dynamic manipulation task where delay management is paramount and showed that transfer across systems with heterogeneous delays and structurally distinct robots is possible. However, the UNN approach only is not sufficient for efficient sim2real transfer, but could be enhanced with other sim2real methods. This work empirically demonstrated the feasibility of our approach on a low dimensional task with constant delay. Future work may investigate the efficiency of our delay-management method on a higher dimensional task and methods to deal with variable delay.

Conclusion

Given the current ecological landscape and the growing significance of environmental concerns in the years to come, several researchers are voicing criticism regarding the unrestrained pursuit of performance and the subsequent excessive utilization of computing power and energy. Moreover, to ensure inclusiveness and avoid a form of centralization in deep reinforcement learning research, it is important to change the current paradigm based on from-scratch learning, which is extremely costly and energy-inefficient. In this thesis, we discussed transfer learning in robotics as a promising avenue for reducing the training costs currently encountered, even with SOTA reinforcement learning algorithms. We argued that enabling robots to share knowledge despite different physical characteristics was a crucial step towards sample-efficient training. As such, we framed our problem as a Cross-Agent Transfer Learning problem and proposed suitable experiments involving multiple robots and tasks to study our problematic. The approaches developed and showcased in this manuscript made concrete contributions to the field of deep reinforcement learning and are further analyzed in the subsequent sections.

Conclusion for the LS-UNN approach

Inspired by Mehdi Mounsif’s work on the Universal Notice Network and transfer learning, we proposed in Chapter 3 a learning method for the emergence of a common representation to support the transfer of knowledge from one robot to another. Our method relies on time-alignment to establish correspondences across robots and minimizes the encoding distance of similar states to create invariance to the morphology. In the experimental part, we demonstrated the effectiveness of this approach through 3 tasks of different nature and transferred corresponding policies across 3 robots of distinct morphology. Compared with Proximal Policy Optimisation, a state-of-the-art learning method, our approach yield a significant sample efficiency boost, while retaining performance. Some transfers result in immediate generalization (zero-shot), and in the worst case significantly reduced the cost of training the target robot. In addition, the bases provide a learning bias which depending on the constraint imposed on the robot during execution of the primitive task(s), can ease learning of downstream tasks. Last but not least, we validated the UNN approach on industrial-grade physical robots, which had not been undertaken in previous work. In addition to the contributions mentioned above, we release the environments created and used for our CATL experiments in the hope that they may serve as a benchmark to compare different transfer approaches on industrial robots with different morphologies, and for tasks of various kinds. To the best of our knowledge, this kind of benchmark has never yet been proposed in the literature.

Perspectives and future works for LS-UNN

Our approach to learn a robot-agnostic space overcomes some of the limitations of the standard UNN, notably the need to define a robot-agnostic space by hand. However, it also introduces its own set of difficulties and questions:

Dealing with over-fitting

As highlighted in Section 4.4.4, LS-UNN suffers from overfitting. Indeed, the UNN module is somehow able to take advantage of the morphology of the robot on which it is learning to continue increasing its reward. As a consequence, the optimized task-strategy becomes less transferable and performance after transfer degrades. While our already proposed solution is effective, it hinders the practicality of this method with additional and time-consuming checkpoint testing. A possible workaround to mitigate over-fitting is to leverage consistent dropout [Hausknecht 22]. Dropout (see Section 2.3.4 for a recall) has proven extremely efficient in many Supervised Learning scenarios, but were rarely used in Reinforcement Learning due to training stability issues. However, consistent dropout were shown recently to provide a stable training, potentially helping RL agents to better generalize. It would be interesting to see whether this type of dropout improves transfer results.

Impact of the type of latent space on transfer and learning.

In this thesis, we restricted ourselves to the study of a latent space obtained with simple VAEs, which we used for training a UNN module and transfers. However, a wide range of more modern alternatives are available with arguably better performance. To name just a few:

- Wasserstein Auto-Encoder [Tolstikhin 17] have proven to outperform the regular VAE on data generation tasks. Given our approach relies on the generative capabilities of the decoder, it could be interesting to explore WAE for transfer.
- Hyperspherical Variational Auto-Encoders [Davidson 18] suggests to use a hyperspherical latent space, employing a von Mises-Fisher prior distribution to more accurately model directional data, such as joint configurations or velocities. Investigating a different and arguably more suited latent-space geometry could unlock more efficient transfers.

Studying these alternatives and their influence on UNN module training (sample-efficiency and asymptotic performance) and the resulting transfer performances is therefore an appealing future research direction.

Building a latent space from unpaired trajectories

Requiring paired and time-aligned trajectories to build the shared latent space can be a practical limitation in some cases. For instance, running the robot to record trajectories can be costly or synchronizing task execution for more complex primitive tasks or robot may prove difficult. Additionally, with the emergence and availability of demonstration datasets for different types of robots on multiple tasks such as Roboturk [Mandlekar 18] or

Robotmimic [Mandlekar 21], it could be of major practical interest to leverage these general purpose, ready-to-use datasets to learn the robot-agnostic feature space. In appendix A, we present some early experiments and results to deal with **paired** but **non-aligned** trajectories using a trajectory-wise, rather than point-wise similarity loss during bases training.

Finally, as learning gait controllers for locomotion tasks is extremely data-hungry, adapting the LS-UNN approach to fit quadrupedal or even humanoid robots is of major interest. A couple of prior works have already attempted transfer for simple ant-like walking robots with promising results.

Conclusion and Prospects for the DA-UNN approach

In order to further validate the UNN approach on a physical robot arm, Chapter 4 tackled the challenge posed by the presence of delay in the robot’s RL control loop. It was observed that this delay adversely affects the performance of an agent trained in simulation when transferred to its physical counterpart. This realization motivated the development of the Delay Aware UNN approach, wherein the delay is randomized during training and supplied as additional information to the task module. Consequently, we obtain a task module capable of adapting to simultaneously to different morphologies and delays. This methodology was validated through simulation and experimentation on two distinct physical robots using a dynamic task. Our experimental findings clearly demonstrated a significant disparity in performance between a delay aware agent and a delay unaware agent, affirming the importance of considering delays in achieving sim2real success.

However, as discussed in Section 5.2.3, our approach may result in sub-optimal policies due to incomplete information in the state. In contrast, many approaches use the information state $I = \{s, a_0, \dots, a_k\}$ to counterbalance the effect of delay. While this approach is theoretically sound and can yield optimal policies in Constant-Delay MDPs, it is not directly transferable as it depends explicitly on k , the delay. But a LSTM-based UNN module could compress I_k into a vector of fixed size no matter what k is, in order to be compatible with a delay range. This could potentially improve results over a simple Multi-Layered Perceptron. Furthermore, from an application point of view it could be interesting to merge our two main contribution, the DA-UNN and the LS-UNN.

Positioning in relation to recent advances in the literature

Some recent advances in the literature need to be discussed and analyzed in order to reposition our contribution. The Isaac-gym simulator [Makoviychuk 21a] enables massive parallel experience collection directly on the GPU which drastically reduces the time needed to train a RL model from scratch. However, the amount of samples to collect is still extremely high as well as the associated energy-consumption. As a result, our approach could complements this high-performance simulator by additionally reducing the amount of interaction needed with the environment, further decreasing the training cost. Other approaches orthogonal to ours such as PalM-E [Driess 23] rely on extremely costly large language models to bring about a common representation suited for task transfer and generalization across morphologies. Although effective, the hardware requirements and massive energy resources required to train and use these kinds of models are prohibitive

for all but a few extremely well funded and equipped laboratories, fueling the race for evermore closed-source, exclusive research and high carbon footprint models. In contrast, we believe that our approach can be applied to reduce resources requirements and meets the Green AI efficiency criteria.

Appendix A

Dealing with misaligned trajectories

In section 4.2.1, we proposed to use time-alignment as a way to pair similar states across robots. While in practice, time-based alignment can be achieved straightforwardly on simple settings, it may not be robust enough when both robots are performing the primitive task at somewhat different speeds, or if the sampling loop does not run at a perfectly fixed rate. To address this issue and guaranty a reasonable alignment and state-pairing regardless of possible desynchronization, we propose to slightly change the bases training procedure. This approach is still under development due to lack of time. Hence, we discuss it in an Appendix.

A.1 Trajectory pairing

This new alignment relies on trajectory pairing rather than state-pairing. As such, agents are not required to execute the primitive task at the same pace, they should simply reach the same state goals. More formally, if we denote by $\pi_{r_1}^{*,\mathcal{T}}$ and $\pi_{r_2}^{*,\mathcal{T}}$ the optimal policies for primitive task \mathcal{T} on respectively, robot r_1 and robot r_2 , then:

$$\tau_{r_1}^{T_1}(s_{g_1}^{\mathcal{T}}) \approx \tau_{r_2}^{T_2}(s_{g_2}^{\mathcal{T}}) \iff s_{g_1}^{\mathcal{T}} = s_{g_2}^{\mathcal{T}} \quad (\text{A.1})$$

where $\tau_{r_i}^T(s_g) = (s_{r_i}^{t_1}, s_{r_i}^{t_2}, \dots, s_{r_i}^{T_i}) \in \mathbb{R}^{T_i \times N_i}$ is a trajectory of length T_i with N_i features, landing on goal state $s_g^{\mathcal{T}}$. In other words, if two trajectories reach the same goal state s_g , they are deemed *equivalent* and can be matched. For instance, if \mathcal{T} is a reaching task, the goal state $s_g^{\mathcal{T}}$ would be the desired end-effector position. This alignment method therefore produces a dataset D of paired and equivalent trajectories such that:

$$D = (\tau_{r_1,n}, \tau_{r_2,n})^{n=1:M} \quad (\text{A.2})$$

where M is the number of trajectories. The training process of the bases given D the dataset of paired trajectories, is relatively similar to the procedure already described in Section 4.2.2. However, as trajectories are not time-aligned but merely equivalent, we can't align the latent space of both robots by minimizing L_{sim} (see Equation 4.4), the loss enforcing proximity in the latent space of two similar and paired states.

A.2 Latent trajectories alignment

A straightforward solution could be to align trajectories using Dynamic Time Warping (DTW) [Müller 07], an algorithm that measures the accumulated cost for the optimal

alignment between two time series. It is a useful tool to compare time series with different lengths as it is robust against shifts or dilatations along the time dimension. It also provides, as a byproduct, the corresponding optimal pairing between points of two time-series with different length. More formally it solves the following optimization problem:

$$\text{DTW}(X, Y) = \min_{\pi \in A(X, Y)} \sqrt{\sum_{(i, j) \in \pi} d(x_i, y_j)} \quad (\text{A.3})$$

where $X = (x_1, \dots, x_{N_x})$ and $Y = (y_1, \dots, y_{N_y})$ are two time-series respectively of length N_x and N_y , $A(X, Y)$ is the set of all possible alignments, π is one possible alignment and $d(x_i, y_j)$ measures the distance between x_i and y_j (the euclidean distance in most cases). Most of the time, this minimal-cost alignment problem is solved using dynamic programming. Reformulating the problem using our specific CATL setting, we consider two joints state trajectories $\tau_{r_1}(s_{g_1}^T) \in \mathbb{R}^{T_1 \times N_1}$ and $\tau_{r_2}(s_{g_2}^T) \in \mathbb{R}^{T_2 \times N_2}$. In general, both robots have a different number of DoFs so $N_1 \neq N_2$. As a result, it is not possible to directly compute the distance (which DTW rely on) between two points on these trajectories due to the dimensionality mismatch. Therefore, $\text{DTW}(\tau_{r_1}^{T_1}(s_{g_1}^T), \tau_{r_2}^{T_2}(s_{g_2}^T))$ is not well-defined and cannot be computed. To overcome this limitation, we propose to minimize the divergence between the latent representations of two equivalent trajectories using Soft Dynamic Time Warping (Soft-DTW) [Cuturi 17], a differentiable formulation of DTW. Soft DTW modifies the original formulation by replacing the min operator with a smooth version:

$$\min^\gamma(a_1, a_2, \dots, a_n) := \begin{cases} \min(a_1, a_2, \dots, a_n) & \text{if } \gamma = 0 \\ -\gamma \log \sum_{i=1}^n e^{-a_i/\gamma} & \text{if } \gamma > 0 \end{cases} \quad (\text{A.4})$$

where γ is a constant used to adjust the smoothing. The optimization becomes:

$$\text{SoftDTW}^\gamma(X, Y) = \min_{\pi \in A(X, Y)}^\gamma \sqrt{\sum_{(i, j) \in \pi} d(x_i, y_j)} \quad (\text{A.5})$$

An immediate consequence of this reformulation is that Soft DTW can be used as a loss function to fit a model, unlike DTW [Cuturi 17]. In our case, it can optimize the bases such that latent projections of two equivalent trajectories are aligned. However, Soft DTW is not a valid divergence because it can be negative and is not minimized when two time-series are equal. These issues are addressed in [Blondel 21] where the authors derived a new divergence D^γ , named soft-DTW divergence and based on Soft DTW. It is define as:

$$D^\gamma(X, Y) = \text{SoftDTW}^\gamma(X, Y) - \frac{1}{2} \text{SoftDTW}^\gamma(X, X) - \frac{1}{2} \text{SoftDTW}^\gamma(Y, Y) \quad (\text{A.6})$$

A.3 Bases training

The new similarity loss introduced in the previous section is straightforward to integrate in the already existing bases training pipeline. We simply need to replace the point-wise euclidean distance by the trajectory-wise SoftDTW divergence. In other words, we now have:

$$L_{sim}(\tau_{r_1}^{T_1}, \tau_{r_2}^{T_2}) = \text{SoftDTW}^\gamma(\tau_{r_1}^{T_1}, \tau_{r_2}^{T_2}) - \frac{1}{2} \text{SoftDTW}^\gamma(\tau_{r_1}^{T_1}, \tau_{r_1}^{T_1}) - \frac{1}{2} \text{SoftDTW}^\gamma(\tau_{r_2}^{T_2}, \tau_{r_2}^{T_2}) \quad (\text{A.7})$$

Additionally, we also remove the L_{CA} cross-alignment term as it requires paired states and can't be adapted for paired trajectories of different length. Finally, the overall objective function using the same notations as Section 4.2.2 is the following:

$$\min_{\theta_{r_1}, \theta_{r_2}, \phi_{r_2}, \phi_{r_1}} \sum_{(x_{r_1}, x_{r_2}) \in D} L_{B_{r_1}} + L_{B_{r_2}} + \delta L_{sim} \quad (\text{A.8})$$

A.4 Experiments

A.4.1 Training setup

We evaluate our alignment approach on the same benchmark as Section 4.3. We enforce misaligned trajectories by setting different task execution speeds. More specifically, taking the UR10 robot as reference, the Panda robot is moving 20% faster and the Braccio robot 40% faster. We generate 2500 equivalent trajectories on a Reaching task across robots and proceed to train a UR10-Panda pair of bases. The final preliminary step is to fit the Braccio bases to align with the Panda ones, following the same procedure as Section 4.2.2 but adapted using SoftDTW. The neural networks architecture is the same as in Section 4.3 and we set δ , the weighting constant for L_{sim} , to 0.0025.

A.4.2 Results

In this section, we present the zero-shot transfer results obtained after training a UNN module on each robots (UR10, Panda and Braccio) and for each tasks (Pick and Place, Peg Insertion and Ball Catching). Performance are measured exactly as in Section 4.4.

Pick and place

Regarding zero-shot transfers performance shown in Table A.1, all transfers except the ones targeting the Panda robot yield a 100% performance success rate. Surprisingly, the Panda robot is not benefiting as much as other robots from transfers, even though it is halfway between the Braccio and the UR10 in terms of speed. Indeed, we would expect UR10 and Braccio transfers to suffer from the relatively large speed difference present while performing the primitive task, resulting in the least successful transfers. However, for a such a simple, non-dynamic task, this kind discrepancies could not be critical.

Source \ Target	Braccio	Panda	UR10
Braccio	100	91	100
Panda	100	100	100
UR10	100	82	100

Table A.1: Performance obtained for UNN agents on the Pick and Place task and zero-shot transfers.

Peg Insertion

For the Peg Insertion task, we observe in Figure A.2 (near) zero-shot generalization in three occasions: UR10 \rightarrow Panda, Panda \rightarrow UR10 and Braccio \rightarrow Panda. The transfers involving the Braccio robot (either as source \rightarrow or target) tend to be less effective. Most notably,

UR10 \rightarrow Braccio results in only a 51% success rate after transfer which is significantly lower than other transfers. Other Braccio related transfers also do not exhibit zero-shot generalization, but are far more beneficial.

Source \ Target	Braccio	Panda	UR10
Braccio	100	75	98
Panda	89	100	100
UR10	51	99	100

Table A.2: Performance obtained for UNN agents on the Peg Insertion task and zero-shot transfers.

Ball Catching

The Ball Catching task is a dynamic task and as such, supposedly very sensitive to trajectories misalignment. Looking at transfer results in Figure A.3, we can see near zero-shot generalization for the UR10 \rightarrow Panda. Other transfers, while still being significantly beneficial, would require further fine-tuning. Surprisingly, the UNN trained on the Panda robot converge to a sub-optimal policy and successfully catch the ball only 87% of the time. In contrast, it can reach a 95% performance when receiving the UR10 UNN, indicating that a near optimal policy do exist in the latent space for the Panda robot, but PPO was not able to find it when training on the Panda, due to a bad seed or other unknow factors.

Source \ Target	Braccio	Panda	UR10
Braccio	97	77	81
Panda	81	87	87
UR10	88	95	98

Table A.3: Performance obtained for UNN agents on the Ball Catcher task and zero-shot transfers.

A.5 Discussion and Perspectives

Overall, transfer performances reported in this preliminary experiment with misaligned trajectories are lower than those reported in Section 4.4. Performance seems to drop as the misalignment increases. Naturally, future work will investigate how efficiently full performance can be recovered when fine-tuning on the target robots. But still, Soft-DTW shows promising results even for early tests with very little hyper-parameters tuning. Moreover, the misalignment considered were voluntarily exaggerated to test the robustness and limitations of this alignment approach. In practice, a de-synchronization of 40 %, as studied for the Braccio robot, is unlikely when working with industrial-grade robots. Therefore, it could be interesting to conduct other experiments with more realistic misalignment to assess the benefit of Soft-DTW for alignment and Cross-Agent Transfer Learning.

Bibliography

- [Ahmed 19] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi & Dale Schuurmans. *Understanding the impact of entropy on policy optimization*. In International conference on machine learning, pages 151–160. PMLR, 2019.
- [Akkaya 19] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribaset *al.* *Solving rubik’s cube with a robot hand*. arXiv preprint arXiv:1910.07113, vol. 10, 2019.
- [Amodei 16] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman & Dan Mané. *Concrete problems in AI safety*. arXiv preprint arXiv:1606.06565, 2016.
- [Andrychowicz 20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Rayet *al.* *Learning dexterous in-hand manipulation*. The International Journal of Robotics Research, vol. 39, no. 1, pages 3–20, 2020.
- [Arndt 20] Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh & Ville Kyrki. *Meta reinforcement learning for sim-to-real domain adaptation*. In 2020 IEEE international conference on robotics and automation (ICRA), pages 2725–2731. IEEE, 2020.
- [Battaglia 18] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner *et al.* *Relational inductive biases, deep learning, and graph networks*. arXiv preprint arXiv:1806.01261, vol. 10, 2018.
- [Beck 22] Nathan Beck, Abhiramon Rajasekharan & Hieu Tran. *Transfer Reinforcement Learning for Differing Action Spaces via Q-Network Representations*. ArXiv, vol. abs/2202.02442, 2022.
- [Behnke 04] Sven Behnke, Anna Egorova, Alexander Gloye, Raúl Rojas & Mark Simon. *Predicting away robot control latency*. In RoboCup 2003: Robot Soccer World Cup VII 7, pages 712–719. Springer, 2004.
- [Bellemare 16] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton & Remi Munos. *Unifying count-based exploration and*

intrinsic motivation. Advances in neural information processing systems, vol. 29, 2016.

- [Bengio 13] Yoshua Bengio, Aaron Courville & Pascal Vincent. *Representation learning: A review and new perspectives*. IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pages 1798–1828, 2013.
- [Bertsekas 00] Dimitri P. Bertsekas. Dynamic programming and optimal control. Vol.1. Numeéro 1 in Athena scientific optimization and computation series. Athena Scientific Publ, Belmont, Mass, 2. ed edition, 2000. OCLC: 833754683.
- [Blondel 21] Mathieu Blondel, Arthur Mensch & Jean-Philippe Vert. *Differentiable divergences between time series*. In International Conference on Artificial Intelligence and Statistics, pages 3853–3861. PMLR, 2021.
- [Bojarski 16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praseoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhanget al. *End to end learning for self-driving cars*. arXiv preprint arXiv:1604.07316, 2016.
- [Bommasani 21] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill et al. *On the opportunities and risks of foundation models*. arXiv preprint arXiv:2108.07258, vol. 10, 2021.
- [Bousmalis 16] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan & D. Erhan. *Domain Separation Networks*. In Neural Information Processing System (NIPS), 2016.
- [Bousmalis 18] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige et al. *Using simulation and domain adaptation to improve efficiency of deep robotic grasping*. In 2018 IEEE international conference on robotics and automation (ICRA), pages 4243–4250. IEEE, 2018.
- [Bouteiller 21] Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal & Jonathan Binas. *Reinforcement learning with random delays*. In International conference on learning representations, 2021.
- [Bro 14] Rasmus Bro & Age K Smilde. *Principal component analysis*. Analytical methods, vol. 6, no. 9, pages 2812–2831, 2014.
- [Brockman 16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang & Wojciech Zaremba. *OpenAI Gym*, 2016.
- [Brown 20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav

- Shyam, Girish Sastry, Amanda Askellet *al.* *Language models are few-shot learners*. Advances in neural information processing systems, vol. 33, pages 1877–1901, 2020.
- [Bócsi 13] Botond Bócsi, Lehel Csató & Jan Peters. *Alignment-based transfer learning for robot models*. In The 2013 International Joint Conference on Neural Networks (IJCNN), pages 1–7, 2013.
- [Campbell 02] Murray Campbell, A Joseph Hoane Jr & Feng-hsiung Hsu. *Deep blue*. Artificial intelligence, vol. 134, no. 1-2, pages 57–83, 2002.
- [Chai 21] Junyi Chai, Hao Zeng, Anming Li & Eric WT Ngai. *Deep learning in computer vision: A critical review of emerging techniques and application scenarios*. Machine Learning with Applications, vol. 6, page 100134, 2021.
- [Chebotar 19] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff & Dieter Fox. *Closing the sim-to-real loop: Adapting simulation randomization with real world experience*. 2019 International Conference on Robotics and Automation (ICRA), pages 8973–8979, 2019.
- [Chen 18] Tao Chen, Adithyavairavan Murali & Abhinav Gupta. *Hardware Conditioned Policies for Multi-Robot Transfer Learning*. page 9355–9366, 2018.
- [Chen 21] Baiming Chen, Mengdi Xu, Liang Li & Ding Zhao. *Delay-aware model-based reinforcement learning for continuous control*. Neurocomputing, vol. 450, pages 119–128, August 2021.
- [Cho 14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk & Yoshua Bengio. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [Chopra 05] Sumit Chopra, Raia Hadsell & Yann LeCun. *Learning a similarity metric discriminatively, with application to face verification*. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), volume 1, pages 539–546. IEEE, 2005.
- [Chowdhery 22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann *et al.* *Palm: Scaling language modeling with pathways*. arXiv preprint arXiv:2204.02311, 2022.
- [Cortes 95] Corinna Cortes & Vladimir Vapnik. *Support-vector networks*. Machine learning, vol. 20, pages 273–297, 1995.

- [Cuturi 17] Marco Cuturi & Mathieu Blondel. *Soft-dtw: a differentiable loss function for time-series*. In International conference on machine learning, pages 894–903. PMLR, 2017.
- [Da Silva 19] Felipe Leno Da Silva & Anna Helena Reali Costa. *A survey on transfer learning for multiagent reinforcement learning systems*. Journal of Artificial Intelligence Research, vol. 64, pages 645–703, 2019.
- [Davidson 18] Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf & Jakub M. Tomczak. *Hyperspherical Variational Auto-Encoders*. 34th Conference on Uncertainty in Artificial Intelligence (UAI-18), 2018.
- [Degraeve 22] Jonas Degraeve, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas *et al.* *Magnetic control of tokamak plasmas through deep reinforcement learning*. Nature, vol. 602, no. 7897, pages 414–419, 2022.
- [Deisenroth 13] Marc Peter Deisenroth, Dieter Fox & Carl Edward Rasmussen. *Gaussian processes for data-efficient learning in robotics and control*. IEEE transactions on pattern analysis and machine intelligence, vol. 37, no. 2, pages 408–423, 2013.
- [Derman 21] Esther Derman, Gal Dalal & Shie Mannor. *Acting in delayed environments with non-stationary markov policies*. International conference on learning representations, 2021.
- [Devin 17] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel & Sergey Levine. *Learning modular neural network policies for multi-task and multi-robot transfer*. pages 2169–2176, 2017.
- [Devlin 18] Jacob Devlin, Ming-Wei Chang, Kenton Lee & Kristina Toutanova. *Bert: Pre-training of deep bidirectional transformers for language understanding*. arXiv preprint arXiv:1810.04805, vol. 10, 2018.
- [Donahue 14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng & Trevor Darrell. *Decaf: A deep convolutional activation feature for generic visual recognition*. In International conference on machine learning (ICLR), pages 647–655. research PMLR, 2014.
- [Driess 23] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yuet *et al.* *Palm-e: An embodied multimodal language model*. arXiv preprint arXiv:2303.03378, 2023.
- [Elfwing 18] Stefan Elfwing, Eiji Uchibe & Kenji Doya. *Sigmoid-weighted linear units for neural network function approximation in reinforcement learning*. Neural Networks, vol. 107, pages 3–11, 2018. Special issue on deep reinforcement learning.

- [Eysenbach 19] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz & Sergey Levine. *Diversity is All You Need: Learning Skills without a Reward Function*. In International Conference on Learning Representations, 2019.
- [(FAIR)† 22] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu *et al.* *Human-level play in the game of Diplomacy by combining language models with strategic reasoning*. *Science*, vol. 378, no. 6624, pages 1067–1074, 2022.
- [Fefferman 16] Charles Fefferman, Sanjoy Mitter & Hariharan Narayanan. *Testing the manifold hypothesis*. *Journal of the American Mathematical Society*, vol. 29, no. 4, pages 983–1049, 2016.
- [Finn 16] Chelsea Finn, Sergey Levine & Pieter Abbeel. *Guided cost learning: Deep inverse optimal control via policy optimization*. In International conference on machine learning, pages 49–58. PMLR, 2016.
- [Finn 17] Chelsea Finn, Pieter Abbeel & Sergey Levine. *Model-agnostic meta-learning for fast adaptation of deep networks*. pages 1126–1135, 2017.
- [Firoiu 18] Vlad Firoiu, Tina Ju & Josh Tenenbaum. *At Human Speed: Deep Reinforcement Learning with Action Delay*. CoRR, vol. abs/1810.07286, 2018.
- [Fukushima 69] Kunihiko Fukushima. *Visual feature extraction by a multilayered network of analog threshold elements*. *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pages 322–333, 1969.
- [Ghadirzadeh 21] Ali Ghadirzadeh, Xi Chen, Petra Poklukar, Chelsea Finn, Marten Bjorkman & Danica Kragic. *Bayesian Meta-Learning for Few-Shot Policy Adaptation Across Robotic Platforms*. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1274–1280, Prague, Czech Republic, September 2021.
- [Girshick 15] Ross Girshick. *Fast r-cnn*. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015.
- [Glorot 11] Xavier Glorot, Antoine Bordes & Yoshua Bengio. *Deep sparse rectifier neural networks*. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.
- [Goodfellow 16] Ian Goodfellow, Yoshua Bengio & Aaron Courville. *Deep learning*. MIT press, 2016.
- [Goodfellow 20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville & Yoshua Bengio. *Generative adversarial networks*. *Communications of the ACM*, vol. 63, no. 11, pages 139–144, 2020.

- [Grigorescu 20] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias & Gigel Macesanu. *A survey of deep learning techniques for autonomous driving*. Journal of Field Robotics, vol. 37, no. 3, pages 362–386, 2020.
- [Gupta 17] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel & Sergey Levine. *Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning*. International Conference Learning Representation (ICLR), vol. 10, 2017.
- [Gupta 18] Madhuri Gupta & Bharat Gupta. *A Comparative Study of Breast Cancer Diagnosis Using Supervised Machine Learning Techniques*. In 2018 Second International Conference on Computing Methodologies and Communication (ICCMC), pages 997–1002, 2018.
- [Haarnoja 18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel & Sergey Levine. *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. International conference on machine learning (ICLR), pages 1861–1870, 2018.
- [Hausknecht 22] Matthew Hausknecht & Nolan Wagener. *Consistent dropout for policy gradient reinforcement learning*. arXiv preprint arXiv:2202.11818, 2022.
- [He 15] Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. In Proceedings of the IEEE international conference on computer vision, pages 1026–1034, 2015.
- [He 16] Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun. *Deep residual learning for image recognition*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
- [He 17] Kaiming He, Georgia Gkioxari, Piotr Dollár & Ross Girshick. *Mask r-cnn*. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.
- [Hejna 20] Donald Hejna, Lerrel Pinto & Pieter Abbeel. *Hierarchically decoupled imitation for morphological transfer*. pages 4159–4171, 2020.
- [Helwa 17] Mohamed K Helwa & Angela P Schoellig. *Multi-robot transfer learning: A dynamical system perspective*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4702–4708, 2017.
- [Higgins 17] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed & Alexander Lerchner. *beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework*. In International Conference Learning Representation (ICLR), 2017.

- [Hinton 12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever & Ruslan R Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580, 2012.
- [Ho 16] Jonathan Ho & Stefano Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, vol. 29, 2016.
- [Hoang 19] Mickel Hoang, Oskar Alija Bihorac & Jacobo Rouces. *Aspect-based sentiment analysis using bert*. In Proceedings of the 22nd nordic conference on computational linguistics, pages 187–196, 2019.
- [Hornik 89] Kurt Hornik, Maxwell Stinchcombe & Halbert White. *Multilayer feedforward networks are universal approximators*. Neural networks, vol. 2, no. 5, pages 359–366, 1989.
- [Hu 19] Yang Hu & G. Montana. *Skill Transfer in Deep Reinforcement Learning under Morphological Heterogeneity*. ArXiv, vol. abs/1908.05265, 2019.
- [Hu 22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang & Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. In International Conference on Learning Representations, 2022.
- [Huang 20] Wenlong Huang, Igor Mordatch & Deepak Pathak. *One policy to control them all: Shared modular policies for agent-agnostic control*. In International Conference on Machine Learning (ICML), pages 4455–4464. research PMLR, 2020.
- [Jackson 21] Lucy Jackson, Steve Eckersley, Pete Senior & Simon Hadfield. *HARL-A: Hardware Agnostic Reinforcement Learning Through Adversarial Selection*. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3499–3505, Prague, Czech Republic, September 2021.
- [Jangir 20] Rishabh Jangir, Guillem Alenya & Carme Torras. *Dynamic cloth manipulation with deep reinforcement learning*. 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 4630–4636, 2020.
- [Jones 09] Susan S Jones. *The development of imitation in infancy*. Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 364, no. 1528, pages 2325–2335, 2009.
- [Judd 08] Charles Hubbard Judd. *The relation of special training and general intelligence*. Educational Review, vol. 36, pages 28–42, 1908.
- [Juliani 18] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar & Danny Lange. *Unity: A General Platform for Intelligent Agents*. ArXiv, vol. abs/1809.02627, 2018.

- [Jumper 21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko *et al.* *Highly accurate protein structure prediction with AlphaFold*. *Nature*, vol. 596, no. 7873, pages 583–589, 2021.
- [Katsikopoulos 03] K.V. Katsikopoulos & S.E. Engelbrecht. *Markov decision processes with delays and asynchronous cost collection*. *IEEE Transactions on Automatic Control*, vol. 48, no. 4, pages 568–574, April 2003.
- [Kim 20] Kuno Kim, Yihong Gu, Jiaming Song, Shengjia Zhao & Stefano Ermon. *Domain Adaptive Imitation Learning*. Rapport technique, July 2020. arXiv:1910.00105.
- [Kingma 14] Diederik P. Kingma & Max Welling. *Auto-Encoding Variational Bayes*. *CoRR*, vol. abs/1312.6114, 2014.
- [Kirillov 23] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo *et al.* *Segment anything*. arXiv preprint arXiv:2304.02643, 2023.
- [Kramer 91] Mark A Kramer. *Nonlinear principal component analysis using autoassociative neural networks*. *AIChE journal*, vol. 37, no. 2, pages 233–243, 1991.
- [Krizhevsky 17] Alex Krizhevsky, Ilya Sutskever & Geoffrey E Hinton. *Imagenet classification with deep convolutional neural networks*. *Communications of the ACM*, vol. 60, no. 6, pages 84–90, 2017.
- [Kurin 21] Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer & Shimon Whiteson. *My body is a cage: the role of morphology in graph-based incompatible control*. *International Conference on Learning Representations (ICLR)*, vol. 10, 2021.
- [Langman 77] Vaughan A Langman. *Cow-calf relationships in giraffe (*Giraffa camelopardalis giraffa*)*. *Zeitschrift für Tierpsychologie*, vol. 43, no. 3, pages 264–286, 1977.
- [LeCun 89] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard & Lawrence Jackel. *Handwritten digit recognition with a back-propagation network*. *Advances in neural information processing systems*, vol. 2, 1989.
- [Lennon 08] Paul Lennon. *Contrastive analysis, error analysis, interlanguage*. *Bielefeld Introduction to Applied Linguistics. A Course Book*. Bielefeld: Aisthesis Verlag, pages 51–60, 2008.
- [Lillicrap 16] Timothy Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra. *Continuous control with deep reinforcement learning*. *CoRR*, vol. abs/1509.02971, 2016.

- [Liu 19] Iou-Jen Liu, Jian Peng & Alexander Schwing. *Knowledge Flow: Improve Upon Your Teachers*. International Conference on Learning Representations (ICLR), 2019.
- [Maas 13] Andrew L Maas, Awni Y Hannun, Andrew Y Nget *al.* *Rectifier nonlinearities improve neural network acoustic models*. In Proc. icml, volume 30, page 3. Atlanta, Georgia, USA, 2013.
- [Makondo 15] Ndivhuwo Makondo, Benjamin Rosman & Osamu Hasegawa. *Knowledge transfer for learning robot models via local procrustes analysis*. In 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), pages 1075–1082, 2015.
- [Makondo 18] Ndivhuwo Makondo, Benjamin Rosman & Osamu Hasegawa. *Accelerating Model Learning with Inter-Robot Knowledge Transfer*. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 2417–2424, Brisbane, QLD, May 2018.
- [Makoviychuk 21a] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa *et al.* *Isaac gym: High performance gpu-based physics simulation for robot learning*. arXiv preprint arXiv:2108.10470, 2021.
- [Makoviychuk 21b] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa *et al.* *Isaac gym: High performance gpu-based physics simulation for robot learning*. arXiv preprint arXiv:2108.10470, 2021.
- [Mandlekar 18] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbayet *al.* *Roboturk: A crowdsourcing platform for robotic skill learning through imitation*. In Conference on Robot Learning, pages 879–893. PMLR, 2018.
- [Mandlekar 21] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu & Roberto Martín-Martín. *What matters in learning from offline human demonstrations for robot manipulation*. arXiv preprint arXiv:2108.03298, 2021.
- [Minsky 69] Marvin Minsky & Seymour Papert. *An introduction to computational geometry*. Cambridge tiass., HIT, vol. 479, page 480, 1969.
- [Mnih 13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra & Martin Riedmiller. *Playing atari with deep reinforcement learning*. arxiv, 2013.
- [Mnih 15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovskiet *al.* *Human-level control*

- through deep reinforcement learning*. nature, vol. 518, no. 7540, pages 529–533, 2015.
- [Moravec 88] Hans Moravec. *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988.
- [Mounsif 19a] Medhi Mounsif, Sebastien Lengagne, Benoit Thuilot & Adouane Lounis. *BAM ! Base Abstracted Modeling with Universal Notice Network : Fast Skill Transfer Between Mobile Manipulators*. July 2019.
- [Mounsif 19b] Mehdi Mounsif, Sebastien Lengagne, Benoit Thuilot & Lounis Adouane. *Universal notice network: Transferable knowledge among agents*. In 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), pages 563–568. IEEE, 2019.
- [Mounsif 20] Mehdi Mounsif. *Exploration of Teacher-Centered and Task-Centered paradigms for efficient transfer of skills between morphologically distinct robots*. Theses, Université Clermont Auvergne [2017-2020], December 2020.
- [Mounsif 23] Mehdi Mounsif, Sebastien Lengagne, Benoit Thuilot & Lounis Adouane. *Universal Notice Networks: Transferring Learned Skills Through a Broad Panel of Applications*. *Journal of Intelligent & Robotic Systems* , 2023.
- [Müller 07] Meinard Müller. *Dynamic time warping*. Information retrieval for music and motion, pages 69–84, 2007.
- [Nath 21] Somjit Nath, Mayank Baranwal & Harshad Khadilkar. *Revisiting state augmentation methods for reinforcement learning with stochastic delays*. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 1346–1355, 2021.
- [Neyshabur 17] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester & Nati Srebro. *Exploring generalization in deep learning*. *Advances in neural information processing systems*, vol. 30, 2017.
- [Ng 00] Andrew Y Ng, Stuart Russell *et al.* *Algorithms for inverse reinforcement learning*. In *Icml*, volume 1, page 2, 2000.
- [OpenAI 23] OpenAI. *GPT-4 Technical Report*. arXiv, 2023.
- [Otter 20] Daniel W Otter, Julian R Medina & Jugal K Kalita. *A survey of the usages of deep learning for natural language processing*. *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pages 604–624, 2020.
- [Pan 10] Sinno Jialin Pan & Qiang Yang. *A survey on transfer learning*. *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pages 1345–1359, 2010.

- [Pathak 17] Deepak Pathak, Pulkit Agrawal, Alexei A Efros & Trevor Darrell. *Curiosity-driven exploration by self-supervised prediction*. In International conference on machine learning, pages 2778–2787. PMLR, 2017.
- [Peng 18a] Xue Bin Peng, Pieter Abbeel, Sergey Levine & Michiel Van de Panne. *Deepmimic: Example-guided deep reinforcement learning of physics-based character skills*. ACM Transactions On Graphics (TOG), vol. 37, no. 4, pages 1–14, 2018.
- [Peng 18b] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba & Pieter Abbeel. *Sim-to-real transfer of robotic control with dynamics randomization*. In 2018 IEEE international conference on robotics and automation (ICRA), pages 3803–3810. IEEE, 2018.
- [Peng 18c] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba & Pieter Abbeel. *Sim-to-real transfer of robotic control with dynamics randomization*. In 2018 IEEE international conference on robotics and automation (ICRA), pages 3803–3810. IEEE, 2018.
- [Radosavovic 23] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik & Koushil Sreenath. *Learning Humanoid Locomotion with Transformers*. arXiv preprint arXiv:2303.03381, 2023.
- [Ramachandran 17] Prajit Ramachandran, Barret Zoph & Quoc V. Le. *Searching for Activation Functions*. CoRR, vol. abs/1710.05941, 2017.
- [Ramstedt 19] S. Ramstedt & C. Pal. *Real-Time Reinforcement Learning*. NeurIPS, 2019.
- [Redmon 16] Joseph Redmon, Santosh Divvala, Ross Girshick & Ali Farhadi. *You only look once: Unified, real-time object detection*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 779–788, 2016.
- [Rombach 22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser & Björn Ommer. *High-resolution image synthesis with latent diffusion models*. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10684–10695, 2022.
- [ROS]
- [Rudin 22] Nikita Rudin, David Hoeller, Marko Bjelonic & Marco Hutter. *Advanced Skills by Learning Locomotion and Local Navigation End-to-End*. In 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2497–2503. IEEE, 2022.
- [Russell 10] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

- [Rusu 15] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu & Raia Hadsell. *Policy distillation*. arXiv preprint arXiv:1511.06295, vol. 10, 2015.
- [Saikat Islam 22] Khan Saikat Islam, Anichur Rahman, Tanoy Debnath & Razaul Karim. *Accurate brain tumor detection using deep convolutional neural network*. Computational and Structural Biotechnology Journal, vol. 20, pages 4733–4745, 2022.
- [Scarselli 08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner & Gabriele Monfardini. *The graph neural network model*. IEEE transactions on neural networks, vol. 20, no. 1, pages 61–80, 2008.
- [Schaal 96] Stefan Schaal. *Learning from Demonstration*. In M.C. Mozer, M. Jordan & T. Petsche, editors, Advances in Neural Information Processing Systems, volume 9. MIT Press, 1996.
- [Schönfeld 19] Edgar Schönfeld, Sayna Ebrahimi, Samarth Sinha, Trevor Darrell & Zeynep Akata. *Generalized Zero-Shot Learning via Aligned Variational Autoencoders*. 2019.
- [Schulman 15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan & Philipp Moritz. *Trust region policy optimization*. International conference on machine learning (ICML), pages 1889–1897, 2015.
- [Schulman 16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan & Pieter Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.
- [Schulman 17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford & Oleg Klimov. *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347, 2017.
- [Schwartz 20] Roy Schwartz, Jesse Dodge, Noah A Smith & Oren Etzioni. *Green ai*. Communications of the ACM, vol. 63, no. 12, pages 54–63, 2020.
- [Shankar 21] Tanmay Shankar, Yixin Lin, Aravind Rajeswaran, Vikash Kumar, Stuart Anderson & Jean Oh. *Translating Robot Skills: Learning Unsupervised Skill Correspondences Across Robots*. In Proceedings of the 39th International Conference on Machine Learning (ICML), PMLR, 2021.
- [Sharif Razavian 14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan & Stefan Carlsson. *CNN features off-the-shelf: an astounding baseline for recognition*. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pages 806–813, 2014.
- [Silver 18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent

- Sifre, Dhharshan Kumaran & Thore Graepel. *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. Science, vol. 362, December 2018.
- [Srivastava 14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research, vol. 15, no. 56, pages 1929–1958, 2014.
- [Starke 17] Sebastian Starke, Norman Hendrich, Dennis Krupke & Jianwei Zhang. *Evolutionary multi-objective inverse kinematics on highly articulated and humanoid robots*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6959–6966, Vancouver, BC, September 2017.
- [Sumers 20] Theodore R Sumers, Mark K Ho & Thomas L Griffiths. *Show or tell? demonstration is more robust to changes in shared perception than explanation*. arXiv preprint arXiv:2012.09035, 2020.
- [Sutton 99] Richard S Sutton, David McAllester, Satinder Singh & Yishay Mansour. *Policy gradient methods for reinforcement learning with function approximation*. Advances in neural information processing systems, vol. 12, 1999.
- [Sutton 18] Richard S Sutton & Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.
- [Szegedy 15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke & Andrew Rabinovich. *Going deeper with convolutions*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.
- [Tang 17] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck & Pieter Abbeel. *# exploration: A study of count-based exploration for deep reinforcement learning*. Advances in neural information processing systems, vol. 30, 2017.
- [Taylor 14] Frederick W Taylor. *Scientific management: reply from Mr. FW Taylor*. The Sociological Review, vol. 7, no. 3, pages 266–269, 1914.
- [Taylor 07] Matthew E Taylor, Peter Stone & Yaxin Liu. *Transfer Learning via Inter-Task Mappings for Temporal Difference Learning*. Journal of Machine Learning Research (JMLR), vol. 8, no. 9, 2007.
- [Todorov 12] Emanuel Todorov, Tom Erez & Yuval Tassa. *Mujoco: A physics engine for model-based control*. In 2012 IEEE/RSJ international conference on intelligent robots and systems, pages 5026–5033. IEEE, 2012.

- [Tolstikhin 17] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly & Bernhard Schölkopf. *Wasserstein auto-encoders*. arXiv preprint arXiv:1711.01558, 2017.
- [Trabucco 22] Brandon Trabucco, Mariano Phielipp & Glen Berseth. *Anymorph: Learning transferable policies by inferring agent morphology*. pages 21677–21691, 2022.
- [Tsounis 20] Vassilios Tsounis, Mitja Alge & Joonho Lee. *DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning*. IEEE Robotics and Automation Letters, vol. 5, 2020.
- [Vaswani 17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin. *Attention is all you need*. Advances in neural information processing systems (NIPS), vol. 30, 2017.
- [Vincent 10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol & Léon Bottou. *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*. Journal of machine learning research, vol. 11, no. 12, 2010.
- [Vogel 16] David Vogel & Audrey Dussutour. *Direct transfer of learned behaviour via cell fusion in non-neural organisms*. Proceedings of the Royal Society B: Biological Sciences, vol. 283, no. 1845, page 20162382, 2016.
- [Walsh 09] Thomas J Walsh, Ali Nouri, Lihong Li & Michael L Littman. *Learning and planning in environments with delayed feedback*. Autonomous Agents and Multi-Agent Systems, vol. 18, pages 83–105, 2009.
- [Wan 20] Michael Wan, Tanmay Gangwani & Jian Peng. *Mutual Information Based Knowledge Transfer Under State-Action Dimension Mismatch*. Conference on Uncertainty in Artificial Intelligence (UAI), 2020.
- [Wang 18] Tingwu Wang, Renjie Liao, Jimmy Ba & Sanja Fidler. *Nervenet: Learning structured policy with graph neural networks*. In International conference on learning representations (ICLR), 2018.
- [Weng 19] Lilian Weng. *Domain Randomization for Sim2Real Transfer*. lilian-weng.github.io, 2019.
- [Wulfmeier 17] Markus Wulfmeier, Ingmar Posner & Pieter Abbeel. *Mutual Alignment Transfer Learning*. In Sergey Levine, Vincent Vanhoucke & Ken Goldberg, editors, Proceedings of the 1st Annual Conference on Robot Learning, volume 78 of *Proceedings of Machine Learning Research*, pages 281–290. PMLR, 13–15 Nov 2017.

- [Zhang 21] Qiang Zhang, Tete Xiao, Alexei A Efros, Lerrel Pinto & Xiaolong Wang. *Learning Cross-Domain Correspondence for Control with Dynamics Cycle-Consistency*. In International Conference on Learning Representations (ICLR), 2021.
- [Zhou 17] Chong Zhou & Randy C Paffenroth. *Anomaly detection with robust deep autoencoders*. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pages 665–674, 2017.
- [Zhu 20] Zhuangdi Zhu, Kaixiang Lin & Jiayu Zhou. *Transfer Learning in Deep Reinforcement Learning: A Survey*. CoRR, vol. abs/2009.07888, 2020.
- [Zhuang 20] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong & Qing He. *A comprehensive survey on transfer learning*. Proceedings of the IEEE, vol. 109, no. 1, pages 43–76, 2020.
- [Ziebart 10] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. Carnegie Mellon University, 2010.
- [Zong 18] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho & Haifeng Chen. *Deep autoencoding gaussian mixture model for unsupervised anomaly detection*. In International conference on learning representations (ICLR), 2018.