UNIVERSITÉ CLERMONT AUVERGNE
ÉCOLE DOCTORALE DES SCIENCES POUR L'INGÉNIEUR DE CLERMONT FERRAND

Thèse

Présentée par

Mehdi Mounsif

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité

Électronique et Systèmes

Exploration of Teacher-Centered and Task-Centered paradigms for efficient transfer of skills between morphologically distinct robots

Soutenue publiquement le 15 Décembre 2020 devant le Jury :

David FILLIAT	Professeur - ENSTA ParisTech	Rapporteur
Olivier STASSE	Directeur de Recherche - LAAS	Rapporteur
Gentiane VENTURE	Professeur - TUAT (Japon)	Examinateur
Sébastien DRUON	Maître de Conférences - LIRMM, UM	Examinateur
Lounis ADOUANE	Professeur - Heudiasyc, UTC	Directeur de thèse
Benoit THUILOT	Maître de Conférences - Institut Pascal, UCA	Co-encadrant
Sébastien LENGAGNE	Maître de Conférences - Institut Pascal, UCA	Co-encadrant

À mon père,
à qui j'aurais voulu montrer
le potentiel professionnalisant des jeux vidéos.

——— *I have to finish this.* ———

Ellie in The Last of Us, Part II - Naughty Dog, 2020

Remerciements

En dépit de ma maîtrise, inédite dans mon entourage (même après trois ans de labeur), des notions d'*acteur-critique*, de divergence de Kullback-Leibler et de l'architecture de ML-Agents, il m'est difficile d'imaginer un scénario d'achèvement de ces travaux de thèse couronné de succès sans le soutien capital de toutes les personnes qui ont pris part, de près ou de loin, à cette aventure et à qui je voudrais ici exprimer ma reconnaissance.

Si ces supports ont été de natures diverses, ils ont pavé la route vers l'achèvement de cette thèse d'un parterre d'herbe chatouilleuse. Sans parvenir à masquer totalement les rares, mais inévitables, aspérités du chemin, le niveau de confort que j'ai perçu excédait de très loin la rugosité et les irrégularités irritantes qu'aurait sans nul doute comporté une trajectoire isolée.

J'aimerais tout d'abord exprimer ma profonde gratitude envers mon directeur de thèse, Lounis ADOUANE. Dans une époque lointaine, ses réponses patientes à mes interrogations sur les systèmes multi-agents ont eu un poids non-négligeable dans mes choix d'orientations. Plus récemment cependant, je voudrais lui signifier mon appréciation pour sa sage vision à long-terme, ses encouragements et ses conseils tactiques en terme de publications qui m'ont permis de valoriser mes travaux. Si je regrette et te présente mes excuses pour t'avoir *emprunté* ton bureau, je suis prêt à te le parier sur le résultat d'un match de basket.

Premier par la chronologie, l'acuité et la pertinence de ses questions lors de la rédaction de nos papiers, je tiens à remercier Benoit THUILOT, qui, après un interrogatoire inoubliable lors de notre première rencontre, m'a permis de rentrer en M1 Robotique, m'ouvrant ainsi le véritable point de départ de cette odyssée. Ainsi, Benoit, si je suspecte que la quantité de sueurs froides et de palpitations que j'ai induite chez toi avec mes robots volants et autres modèles de jeux vidéo dépasse de très loin le quota autorisé, je te remercie ta pédagogie, ton humour et ton bon sens qui ont été salvateurs à plus d'un égard.

Finalement, une mention particulière à Sébastien LENGAGNE, troisième membre de l'équipe encadrante (la team des Boyz, comme je m'y référais dans le secret du bureau 3106). Au cours de ces trois ans au front, Sébastien à vaillamment endossé le rôle d'encadrant de terrain. N'hésitant jamais à me lancer à l'assaut des territoires inexplorés et dangereux du transfert entre agents de morphologie différentes, guidé par une connaissance intime des réflexes de la communauté robotique, il m'a permis de mettre en place des plans d'expériences ambitieux sur l'Universal Notice Network qui feraient pâlir d'envie les clusters de calcul des GAFAM. Je voudrais te témoigner Sébastien, ma profonde gratitude: tu as été un encadrant d'une rare gentillesse, attentif et à l'écoute. Malgré nos épisodiques divergences sur le mode de programmation des robots, dont Benoit pourra certainement attester, tu as supporté et encouragé la grande majorité de mes initiatives scientifiques en trouvant un délicat équilibre entre le pilotage et l'autonomie. Je suis tout à fait conscient de la chance que j'ai eu.

J'aimerais aussi adresser mes remerciements aux membres du jury, Gentiane VENTURE, Olivier STASSE, David FILLIAT et Sébastien DRUON dont les questions sagaces, la justesse scientifique, l'ouverture d'esprit et l'amabilité ont conféré une aura particulière au souvenir que je garde du 15 décembre 2020.

Plus familièrement maintenant, je dédie mon prochain big-up aux locataires du bureau 3016 avec qui j'ai partagé, somme toute, une proportion non-négligeable de mon temps éveillé de ces dernières années. Pendant ces moments, nous avons partagé les joies et les peines communes des doctorants, l'effusion des nouvelles idées et l'émulation scientifique, des courses à la deadline, des meetings MAACS, des pétitions contre le bruit des machines de l'étage du dessus ainsi que des discussions animées, parfois profondes et parfois un peu moins, mais systématiquement rythmées par le bruits de la chasse d'eau des sanitaires d'en face. Mention particulière à Nadhir et sa douceur, à Lobna, à Kamel, à Jojo la Fripouille. À Dimia aussi, sans qui je n'aurais jamais pu m'inscrire en deuxième année, faute de compétences techniques suffisantes pour le scanner du deuxième étage.

J'adresse aussi une pensée transpirante au club de boxe anglaise de Clermont Boxe St-Jacques, dont l'affiche à longtemps trôné sur la porte du bureau, indiquant aux visiteurs, grâce à une inscription manuscrite de Dimia, ma victoire du 30 Novembre 2018. Merci à Adel et Bénédicte d'avoir joué un rôle si structurant pendant ces années et de m'avoir offert un environnement dans lequel j'ai pu me défouler, progresser comme boxeur et pratiquer mes talents d'orateurs en tant que coach.

Pour conclure, je destine le dernier paragraphe de ces remerciements à mon premier cercle, plus restreint, mais dont le soutient fut incontournable au cours de cette thèse. À ma mère et ma soeur, inconditionnelles dans leur amour stratosphérique et leur affection et que j'espère rendre aussi fières de moi que je le suis d'elles. À Héloïse également, Capitaine de mon navire, qui attendait la conclusion de cette thèse et notre départ vers la vie de JCD (Jeunes Cadres Dynamiques) avec encore plus d'impatience que moi.

Afin de ne pas diminuer excessivement la patience du lecteur qui doit encore traverser plusieurs dizaines de pages pour arriver au bout de ce manuscrit, j'abrégerais ces remerciements en saluant tous ceux, proches, amis et famille, que je n'ai pas directement cités ici, mais dont l'affection restera liée dans mon esprit à ces trois années inoubliables.

Résumé

Récemment, il a été possible d'observer l'accélération du déploiement de robots dans des domaines dépassant l'habituel cadre industriel et manufacturier. Cependant, pour la majorité des tâches autonomes, la définition d'un modèle analytique ou la recherche d'une solution optimale requiert des ressources rarement accessibles en temps-réel, favorisant par conséquent des techniques basées sur l'apprentissage. Ces dernières, présentant l'avantage de ne pas nécessiter de modèle ainsi que de présenter un temps de calcul en exécution relativement constant, permettent d'appréhender des configurations et tâches hautement complexes. Les techniques basées sur les données font cependant état de temps d'entraînement considérables, nécessitant fréquemment des millions d'exemples et d'interaction avec leur environnement pour construire des politiques de contrôle admissibles. Le transfert de connaissance entre modèles est crucial pour le déploiement à grande échelle des méthodes d'apprentissage et bien que des stratégies de transmission aient été au coeur des récentes préoccupations, elles sont essentiellement dirigées vers les domaines de vision ou de compréhension du langage et ne sont pas directement applicables à des problématiques de transfert de compétences entre robots présentant des structures cinématique différentes. Les travaux présentés dans ce manuscrit de thèse se focalisent précisément sur ce point et visent à déterminer dans quelle mesure la compréhension entre deux entités morphologiquement distinctes est possible. Cette question est explorée *via* l'introduction de deux paradigmes distincts: Task-Centered et Teacher-Centered. La famille de techniques dite Task-Centered est basée sur l'idée de la séparation du savoir-faire relatif à une tâche des stratégies de contrôle du robot. A la manière d'une notice d'instruction, un tel noyau indépendant peut par conséquent être passé à d'autres robots de morphologie différentes et idéalement rendre possible la réalisation de la tâche par ce nouvel agent. Dans ce contexte, plusieurs procédures de création de ce noyau sont proposées et évaluées sur un large panel d'environnements simulés. Cependant, en dépit des perspectives attractives de cette formulation, le caractère "one-size-fits-all" des techniques Task-Centered n'est pas exempte de limitations qui sont extensivement discutées. C'est dans ce contexte que les approches Teacher-Centered sont introduites. Poursuivant le même objectif, ces démarches innovantes font intervenir un agent expert à partir duquel le savoir relatif à la tâche doit être distillé dans l'agent cible. Pour ce faire, une métrique originale est utilisée pour contourner la différence de structure entre l'agent cible et l'agent expert et permettre, malgré cette distinction, la rétro-propagation de l'erreur afin d'optimiser l'agent.

Abstract

Recently, it has been possible to observe the acceleration of robot deployment in domains beyond the usual industrial and manufacturing framework. However, for the majority of autonomous tasks, the definition of an analytical model or the search for an optimal (or acceptable) solution requires resources that are seldom available in real-time, thus favoring learning-based techniques. Indeed, learned models present the advantage of being model-free as well as having a constant execution time, consequently enabling the realization of highly complex trajectories and tasks. Data-driven techniques, however, are hindered by considerable training time, frequently requiring millions of examples and interactions with their environment to build acceptable control policies. As such, knowledge transfer, also known as transfer learning, between models is crucial for large-scale deployment of learned policies. Although transmission strategies have been the focus of recent concerns, they are mainly directed towards the fields of vision or language understanding and are not directly applicable to control environments where skill transfer is likely to happen between robots with different kinematic structures. The works presented in this thesis manuscript focus precisely on this point and aims at determining to what extent understanding between two morphologically distinct entities is possible. This question is explored through the introduction of two distinct paradigms: Task-Centered and Teacher-Centered. The Task-Centered family of techniques is based on the idea of the separation of task-related know-how from robot control policy. Such an independent kernel can therefore be passed on to other robots of different morphology and ideally make it possible for the new agent to perform the task. In this context, several blueprints for creating this kernel are proposed and evaluated on a wide range of simulated environments. However, despite the attractive prospects of this formulation, the "one-size-fits-all" character of Task-Centered techniques is not free of limitations which are extensively discussed. It is in this context that Teacher-Centered approaches are introduced. Pursuing the same objective, these innovative procedure involve an expert agent from which the knowledge related to the task must be distilled into the target agent. To do this, an original metric is used to circumvent the structural differences between the target agent and the expert agent and allow, despite this distinction, the error to be back-propagated in order to optimize the agent.

Acronyms

AE	AutoEncoder
BAM	Base Abstracted Modelling
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CV	Computer Vision
DC	Differentiable Collision
FFNN	Feed Forward Neural Network
GAN	Generative Adversarial Network
ISV	Intermediate State Variable
KL	Kullback-Leibler
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Networks
NLP	Natural Language Processing
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SAC	Soft Actor-Critic
SL	Supervised Learning
TAC	TAsk-Centered
TEC	TEacher-Centered
TSL	Task-Specific Loss
UNN	Universal Notice Network
VAE	Variational AutoEncoder
WGAN-GP	Watterstein Generative Adversarial Neural with Gradient Penalty

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Opening

Within the animal reign, there exists a wide variety of behaviours regarding the relation between an adult individual and its children. For instance, most invertebrates leave their eggs alone once laid, relying on numbers to ensure that a viable size of newborn survives. As opposed to this, mammals are among the species that take the most care of their offspring as adults effectively provide food and protect their children until they are strong enough to do so for themselves. Within the invertebrate family, octopuses and squids are famous for their particularly high brain/body ratio, their ability to use tools and learn through observation and play. These specific features are among those who enabled mankind to move from a hunter-gatherer existence to space exploration. Thus, among other reasons, it is possible to consider that the absence of knowledge transmission between generations heavily impacts the development of a specie as a whole.

Throughout their history, humans have taken the measure of how crucial knowledge transmission is, to the point that it is institutionalized (in schools, universities, institutes, for instance) in our societies. Although it is a very common and well-spread custom, transferring knowledge from one individual to another one or to another group is still, depending on the complexity, seldom immediate and requires repetition and training over a non-negligible period of time to be effectively assimilated.

Even though the transfer process within human communities frequently relies on the physical presence and demonstration of an expert individual (a teacher), the usage of a intermediate support, such as books, podcasts or videos is also widely accepted. The diversity of such intermediate mediums illustrates that there exist several conceivable approaches for transfer: from a personalized, individually-targeted process (a one-on-one sport class) to an industrial, mass-directed approach (printing a widely distributed ideological pamphlet).

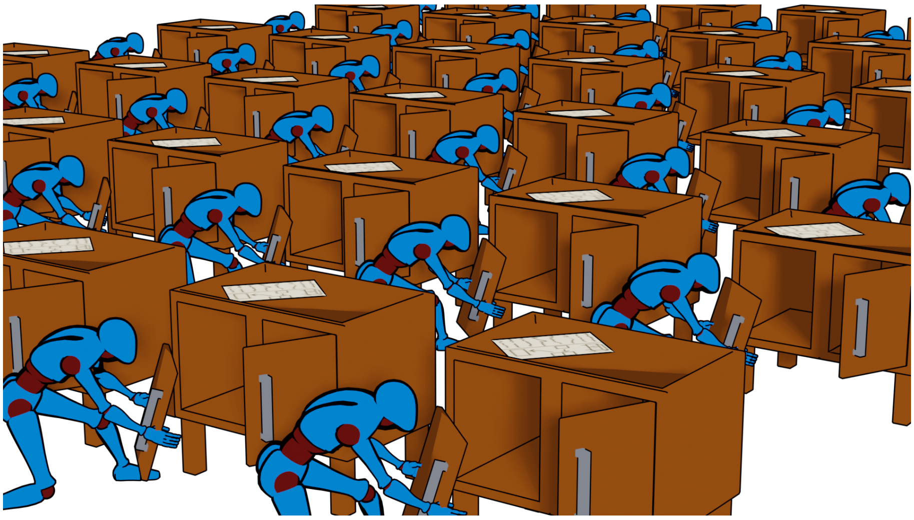


FIG. 1.1. A mass directed, widely distributed, list of instructions to assemble a piece of furniture

A supplementary challenge in the transfer process is related to the diversity of the knowledge shared, that in a number of areas, is expected to yield a perfect copy while it relates more to a desired result in others. For instance, when assembling furniture, the knowledge written by an expert on the notice is expected to allow the customer to setup the appliance flawlessly while the techniques given by an experimented climber to newcomers is broader and only aim at giving them the possibility to climb in a more efficient way. The two previous examples also highlight an important difference which relates to the degree of control on which the transfer focuses. Specifically, the furniture case deals with high-level instructions ("nail panel B to door A") while the climbing environment presents situations likely to call for lower-level considerations ("Push on your left leg to stabilize your grip") that depend directly on the attributes of the transfer target.

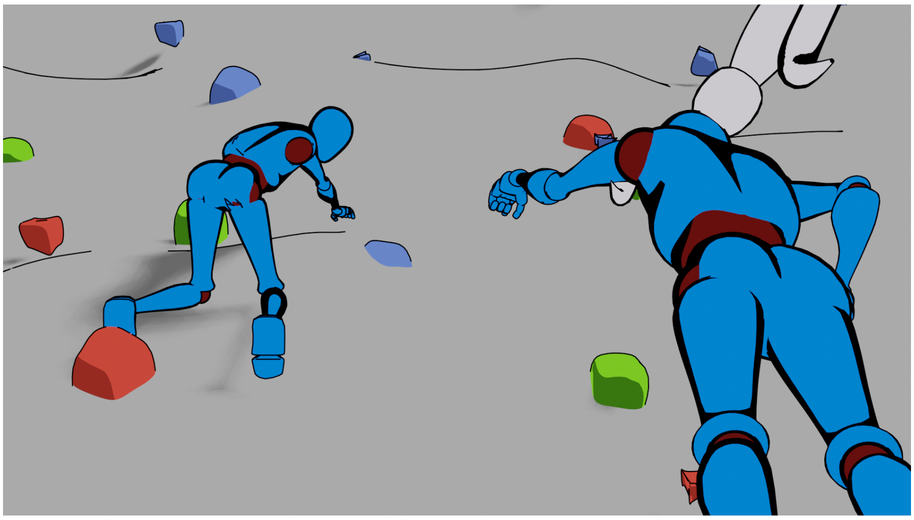


FIG. 1.2. Personalized advice in a bouldering configuration: the experimented climber distills its knowledge in the student accomplishing the task

As recent progresses in robotics and artificial intelligence let us envision a future where robot presence and activity will be ubiquitous, it is likely that, fueled by economics, cultural background, and design choices, human creativity is highly liable to design robots of various forms and shapes. As a result, these robots will certainly present a wide range of sensors and actuators to accomplish their tasks. However, as opposed to information processing domains such as vision or audio which can depend on a single unified representation (pixels and soundwave, respectively), robot parts may present different kinematic structures, power and overall different mechanic functions. Consequently, while it may appear viable to transfer a vision model, provided the good normalization, it is less straightforward to share the same control structure.

While efforts in robotics have resulted in efficient analytical formulation for numerous tasks and problems, the vast majority of the functions that humanity expects robots to perform is likely to rely on learning-based control, as defining an accurate model for these actions can be difficult. However, as of today, even the most recent algorithms are still experience-greedy and require millions of examples and interactions with an environment to produce acceptable policies for a single agent, thus yielding daunting considerations in

terms of computation if this strategy was to be applied to each different robot.
Consequently, the capacity to transfer skills from one agent to another, notwithstanding their distinct physical structure, is a crucial and essential step in the path of integrating robots in our day-to-day lives. This work consequently proposes novel techniques to face this issue. These approaches can be classified into two main families based on whether they fit in the furniture example (TAsk-Centered), displayed in Figure 1.1 or climbing configuration (TEacher-Centered) as shown in Figure 1.2.

1.2 Goals

This work challenges the current transfer learning paradigm and aims at showing that it is possible to transfer knowledge from an agent to a kinematically different one. Metaphorically, this work can be seen as an echo to George Berkeley's *immaterialism* (6) theory which denies the existence of material substance outside of the mind of perceiver. The methods proposed in this work rotate this interrogation towards knowledge and investigate whether the skills and knowledge to successfully complete a task are tied to the morphology which learned to do so. To sum up, this thesis explores the following hypothesis: is it possible to transfer knowledge, that is the expertise or control strategy in a given environment, from one agent to another despite their potential morphology difference ? And, if so:

- how to cope up with the action space dimensional differences ?
- can a *common state* be defined ?
- to what extend can knowledge be distinct from the body ?

1.3 Proposed Approaches

This thesis focuses on two distinct ideas for transferring knowledge: TAsk-Centered (TAC) and TEacher-Centered (TEC). Fueling the TAsk-Centered approaches is the idea of notices, similar to the ones used by non-professional humans to quickly be able to produce an object (for instance, setting up a piece of furniture) without having specifically been trained on this type of task. In these cases, the furniture producer writes down a set of instructions that should enable another person to reach its goal, that is, assembling the newly acquired furniture. In this configuration, the producer does not have access to the customers physical abilities, but makes the assumption of basic motor skills that would allow him to comply with the notice's current set of instructions. In this view, the TAsk-Centered methods first aim at constructing a notice module, independent from the acting agent morphology. Once this notice module is available, it can be passed to other agents that would then perform the task. This approach is schematised in Figure 1.3 that displays an example on how two systems similarly tasked can present different strategies to solve the given configuration.

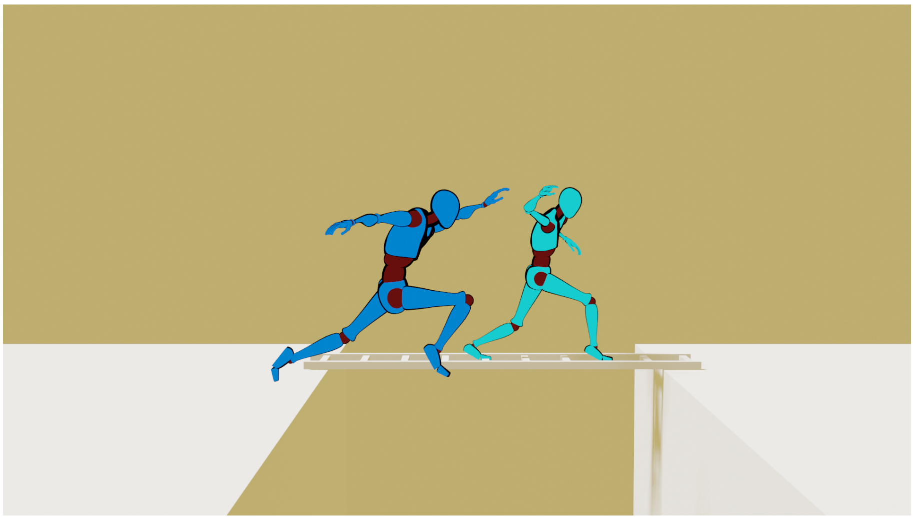


FIG. 1.3. TAsk-Centered (TAC) approach with the instruction "Cross the gap" can be answered differently given the capabilities of the current agent.

It is also possible to consider TEacher-Centered approaches. In this view, instead of having a segmentation between the knowledge and the agent morphology as in TAsk-Centered cases, the task knowledge is tied to an agent, called the teacher or the expert. The goal of TEacher-Centered techniques, displayed in Figure 1.4, is to provide ways to distill this ability into another student agent of potentially different morphology. This configuration relates to very common settings in human civilization. Indeed, there exist numerous examples where an untrained individual can benefit from the knowledge of an expert, thus shortening the time needed to reach mastery.

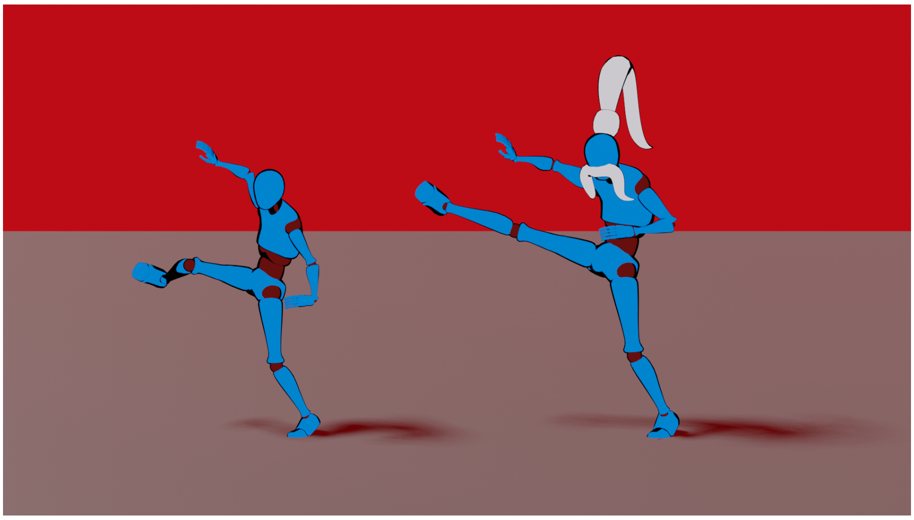


FIG. 1.4. TEacher-Centered approach to Kung-Fu. In this configuration, knowledge from the teacher agent (right) is distilled to the learner agent (left)

1.4 Contributions

As this report proposes several approaches, a novel approach classification was devised. Indeed, TAsk-Centered and TEacher-Centered techniques are two newly defined families of transfer-learning approaches that usefully draw a line between the novel methods proposed in this work.

More precisely, in the TAsk-Centered approach, we propose the Universal Notice Network (UNN) (84; 85) a stage-wise deep reinforcement learning based technique that creates a task knowledge module that can then be passed to different agents. Doing so enables to segment the task logic, that should be common to all agents, from the control parameters inherent to each robot. This technique yields very interesting results both for the learning part as well as the transfer, as it enables other agents to reach performances level similar to the expert in a fraction of the training time. Besides, the segmentation pipeline involved in this construction presents unexpected, yet highly appealing features, with respect to agent control parameters. Specifically, in the *pre-training* phase, it is easier to induce biased behaviors in the agent low-level control. These behaviors can then importantly ease the learning process in the target task and also contribute to prevent undesired or unsuitable actions downstream.

For the TEacher-Centered framework, this report investigates innovative unsupervised techniques to distill the teacher knowledge into a student agent. As opposed to TAsk-Centered techniques, where the transfer of knowledge is horizontal, these methods are based on a clear hierarchy and the student quality is directly submitted to the teacher's, which was less straightforward for TAsk-Centered framework. Nevertheless, TEacher-Centered methods present several strong advantages. Notably, this form of transfer is faster, and does not require student-environment interactions, thus lessening the cost of transfer. Furthermore, TEacher-Centered methods embed more freedom in the objective function design, ultimately offering a very wide application spectrum due to their enhanced flexibility.

1.5 Manuscript outline

The remainder of the thesis is organized as follows: Chapter 2 provides an overview of the concepts and theoretical notions required. It introduces the recent paradigm machine learning and quickly focuses on deep learning. After laying out the mechanisms of neural networks, it goes over the most popular deep learning applications. After an outline of supervised learning, it examines thoroughly reinforcement learning as it is heavily relied upon in the next chapters. Eventually, Chapter 2 looks at unsupervised learning and knowledge representation by presenting attractive frameworks: Generative Adversarial Networks and Variational AutoEncoders that are also used in the works presented in this report.

Chapter 3 sums up current state of the art techniques, modern trends in transfer learning and highlights their major advantages as well as their limitations, thus motivating the proposed transfer-learning approaches.

In Chapter 4, the Universal Notice Network, a TAsk-Centered transfer learning approach is introduced. This chapter details its main features and presents a series of experiments in multiple use cases to underline the applicability of this method.

TEacher-Centered methods are at the center of Chapter 5, which develops the essential aspects of the CoachGAN and Task-Specific Loss (TSL) methods and lays down the results of various trials designed to underline the potentiality of these techniques.

Eventually, as could be expected, Chapter 6 sets up the stage for a thorough comparison between all developed methods in this works, demonstrating multiple cases of transfer between a series of robots on a common task, ultimately leading to an analysis and conclusion on transfer-learning in the control-tasks paradigms.

Chapter 2

Neural Networks and Learning Frameworks: Theoretical Background

Contents

As introduced in the previous chapter, this work aims at presenting various techniques and concepts for the transfer of knowledge between heterogeneously shaped agents. To do so, it relies extensively on the numerical methods tied with neural networks. Consequently, this chapter presents the theoretical background required for the work addressed in this PhD thesis. It begins by introducing the general principle and motivation of Machine Learning. Then it focuses on Neural Networks concepts and main architectures. Once these building blocks are in place, it goes over the principal learning paradigms, supervised learning, reinforcement learning and finally unsupervised learning techniques and knowledge representation.

2.1 Machine Learning

The idea of intelligent machines can be found in many cultures, some of them reporting back to the antiquity, in the form of thought-capable artificial beings (113). Closer to the modern times, various fictions explored this concept further, such as Frank Baum's *Wizard of Oz*, Mary Shelley's *Frankeinsten* or Karel Capek's *R.U.R.* (143). The field of Artificial Intelligence draws from these ideas, but even more from the famous Computing Machinery and Intelligence from Alan Turing (17; 138) in 1950.

Artificial Intelligence (AI) is the science that investigates *the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings* (18). The beginnings of AI research are commonly associated with the Dartmouth 1956 computer science conference which reunited some of the brightest minds of the era and was expected to lay out the main principles that would ultimately lead to reproduce the human mind *in silico*. Initially, AI was dominated by rationalist ideas, which, inspired by thinkers such as Descartes, Spinoza or Leibniz, consider that our perceptions are fallible and that only reason can be a reliable guide. In practice, these ideas lead to knowledge engineering implementations that require formal description of a problem. Despite impressive first successes, such as the General Problem Solver or the $\mathbf{A}\star$ algorithm (113), these approaches proved inefficient at best for many complex tasks that can be considered intuitive, such as recognizing a face or spoken words.

Machine Learning (ML), in contrast, is inspired by empiricists such as Locke, Berkeley or Hume, which refute the idea that only reason is reliable and would preferably use their perceptions to guide them in the world (23; 113). Consequently, ML can be loosely defined as gathering knowledge from experience. Specifically, the main idea fueling the ML paradigm is that, instead of hard-coding knowledge within an AI system, it is possible to find patterns in raw data and consequently use these patterns to solve various challenging tasks that would require tedious and particularly complex algorithms otherwise. As of today, ML algorithms are able to segment images and generate new human faces with a strong accuracy or even predict the best moves in Go (126; 127), a high-dimensional game that remained out of reach for many years.

2.2 Neural Networks

Neural networks, also known as artificial neural networks, are the results of attemps to find mathematical representations of how information was processed in biological systems. It is currently a highly popular technique in machine learning that is used for pattern recognition by extracting statistical features (8; 39). As hinted by their designation, neural networks can be seen as a combination of single neurons, recalling their biological counterpart. The next sections present an overview of these systems, going from a single neuron as a computational unit to more complex architectures.

2.2.1 Neurons

In the field of deep learning, the term neuron n is commonly used to refer to a computational unit that takes as input a vector $X \in \mathbb{R}^N$, with $N \in \mathbb{N}^+$, and computes an output value $a \in \mathbb{R}$ given a bias $b \in \mathbb{R}$, a vector of weights $W \in \mathbb{R}^N$ and an activation function g. More precisely, the neuron computes the pre-activation value a as the weighted sum of the input vector and adds a bias:

$$a = W^T X + b = \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

The result of the operation described in Equation 2.1 goes through an activation function a that returns the final value o for this specific unit:

$$o = g(a) \tag{2.2}$$

A neuron is considered deterministic if its output o is a function such as:

$$o = g(a) \text{ where } g{:}\mathbb{R} \to \varepsilon \subset \mathbb{R} \tag{2.3}$$

Similarly, a neuron is labelled stochastic if its output o is sampled from a density function depending on the pre-activation function

$$o \sim g(a) \tag{2.4}$$

Figure 2.1 shows a scheme of this computational unit and the operation it performs.

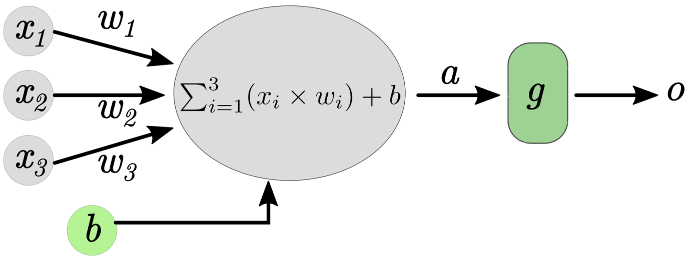


FIG. 2.1. The neuron is the basic computational unit in a neural network.

A neural network, also known as Multi-Layer Perceptron (MLP), is created when several layers of multiple neuron units followed by non-linear activation functions are assembled. In general, MLPs are able to learn complex mappings and, as demonstrated by the Universal Approximation Theorem, a neural network with three layers wide enough can in theory approximate any function (19; 45; 47).

2.2.2 Feed forward neural networks

Modern neural network architectures are defined as Feed Forward Neural Networks (FFNN). The word feedforward stems from the fact that information flows through the various layers, producing intermediate representations to be used by the next layer of perceptrons, until the final prediction. More formally, it is possible to describe these architectures as a directed acyclic (no cycle and no self-connections) graph that composes several functions. These functions are in fact each of the network layers, usually referred to as input, hidden and output, for the first layer, the intermediate layers and the last layer respectively, see Figure 2.2. Beside FFNN, there exists alternative designs, such as Recurrent Neural Network (RNN) that prove useful when considering time-series processing. These networks usually rely on specific neurons to function, such as the Long Short Term Memory (LSTM) and will not be detailed in this report.

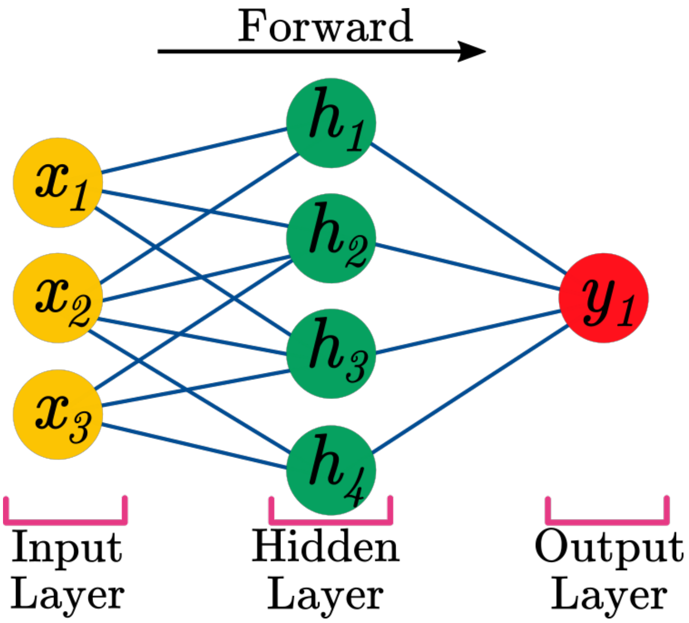
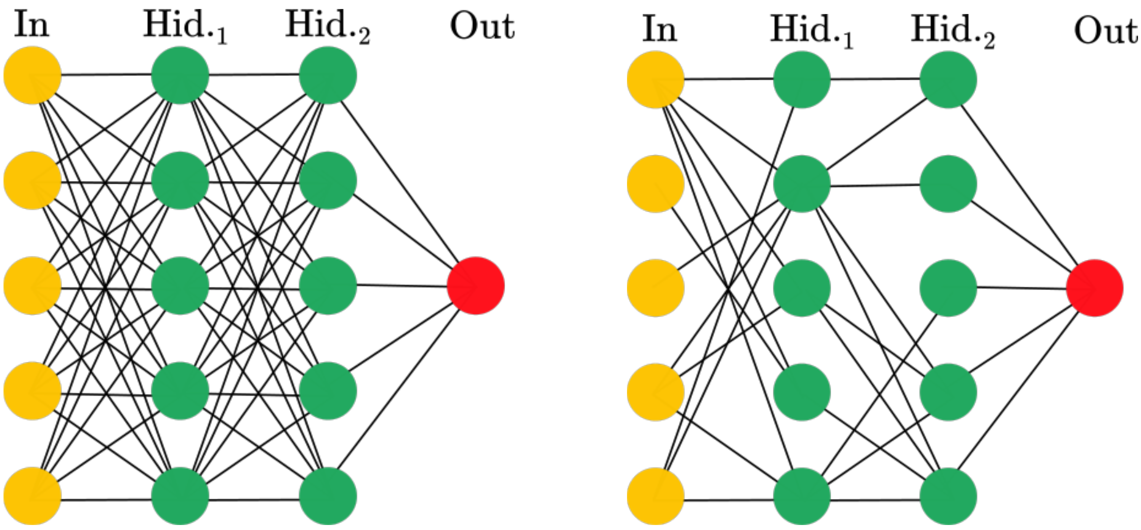


FIG. 2.2. Feedforward neural network: vectors are sequentially processed by each layer, first applying the layer weight parameters and then the non-linearity activation function. The last layer returns a value that is usually used to compute the loss.

Figure 2.2 represents a simple feedforward neural network architecture. Specifically, the network displayed contains 3 units in its input layer (yellow), 4 in its unique hidden layer (green) and finally 1 in the output layer (red).



(a) Fully connected layer.

(b) Partially connected layer.

FIG. 2.3. Layer connections.

2.2.2.1 Layers

The neural network displayed in Figure 2.3a has only fully-connected layers. Recently, it has become more common to create architectures relying on partially connected layers. In particular, the Convolutional Neural Networks (CNN) architecture that uses convolutional kernels as trainable weights has generated considerable interest in recent years. Differences between these layers are displayed in Figure 2.3.

A layer is called fully-connected if each unit of a given layer l is connected to each unit of the previous layer $l-1$. In this case, the activations can be computed as:

$$\forall i \in \{1 \ldots N_l\},\ \ a_i^l = \sum_{j=1}^{N_{l-1}} w_{i,j}^l o_j^l + b_i^l \tag{2.5}$$

and the outputs are given by:

$$\forall i \in \{1 \ldots N_l\},\ o_i^l = g_l(a_i^l) \tag{2.6}$$

where N_l is the number of units in the layer l, $w_{i,j}^l$ being the weight between the jth neuron of the layer $l-1$ and the ith neuron of the layer l and b_i^l is the bias of the ith neuron in layer l.

In contrast, a layer l is considered partially connected if it contains at least one unit that is not connected to the previous layer $l-1$. It is possible to devise the activation and output of a partially-connected layer by setting to 0 the connection weight between two unconnected neurons. A particularly common instance of partially-connected layer is the convolutional layer, most commonly used in image processing. Beside the obvious decrease in computational load when relying on partially connected layers, convolutional layers also offer, by design, a larger perception field, which allows them to detect hierarchical patterns, thus justifying their usage when considering high-dimensional structured inputs.

2.2.2.2 Activation functions

The pattern recognition ability of neural networks is rooted in the presence of non-linear activation functions. Indeed, a neural network with many layers featuring solely linear activation functions is equivalent to a single-layer linear neural network, which limits its expressivity and flexibility (39). This section introduces the most common activation functions in recent literature.

Sigmoid

The sigmoid function was particularly popular in the early days of neural networks due to the fact that it recalls the behaviour of a biological neuron. It outputs values ranging from 0 to 1 and is a monotonically increasing function. Mathematically, it can be put as:

$$g_{\text{sigmoid}}(x) = \sigma(x) = (1 + e^{-x})^{-1} \tag{2.7}$$

One of the main drawbacks of this activation function is that it saturates for considerable values and, as can be seen in Figure 2.4, its derivative never exceeds 0.25, which leads to shrinking gradient signals when several layers using this activation function are composed. This consequently prevents the first layers of a neural network to be efficiently modified and thus, to learn. This issue is common when training neural networks and is usually referred to as gradient vanishing. The Sigmoid function is nevertheless still in usage, particularly at the end of neural networks with a single output, where the resulting value can be understood as probabilities.

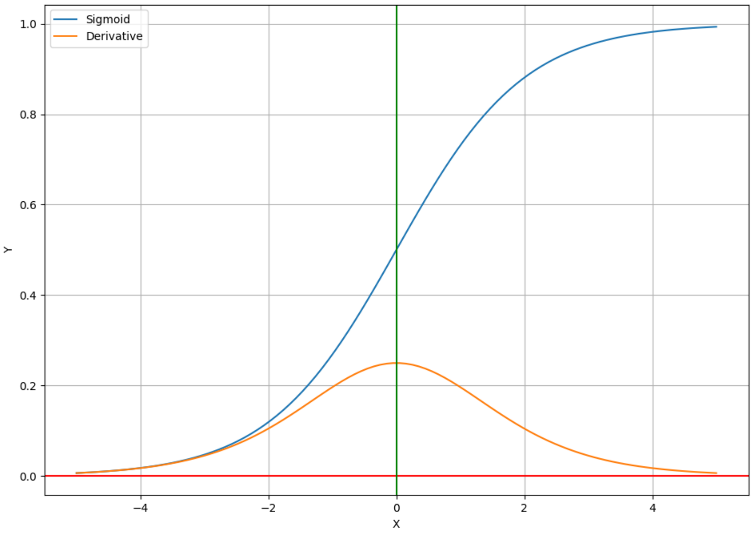


FIG. 2.4. The sigmoid function and its derivative.

TanH

The hyperbolic tangent, or TanH, activation function closely resembles the sigmoid function but presents a larger range of $[-1, 1]$ and is symmetric with respect to the origin. While it is also liable to saturate to important values, it nevertheless provides more gradients than the sigmoid, thus being less likely to cause gradient vanishing. This non-linearity is particularly frequent in reinforcement learning architectures. Mathematically, it can be described by:

$$g_{\text{tanh}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.8}$$

Figure 2.5 displays the hyperbolic tangent and its derivative.

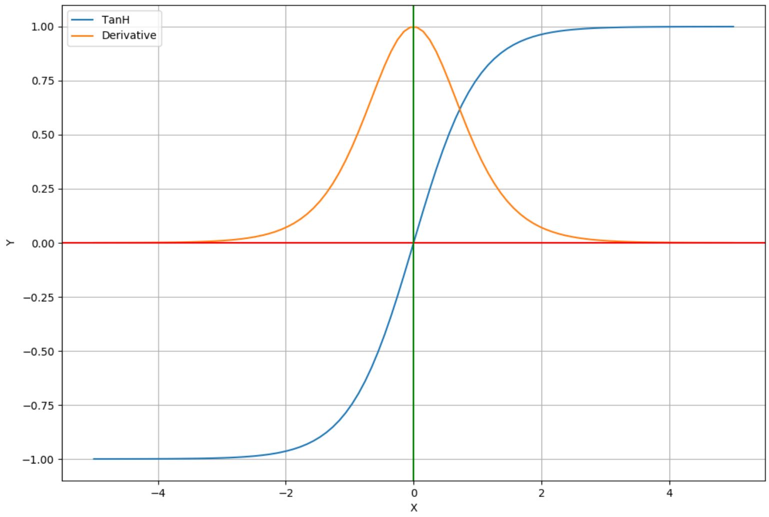


FIG. 2.5. The TanH function and its derivative.

ReLU

The Rectified Linear Unit (ReLU) function, is a linear function for the positive domain but returns 0 when its input is negative. Consequently, the gradient for this function is either 0 or 1, effectively preventing the gradient vanishing issue, as shown in Figure 2.6. It is computationally efficient and has gained a lot of popularity over the past years. The fact that this function does not saturate improves networks performances in terms of convergence rate and accuracy for discriminative settings (39). On the downside, it has been shown that ReLU activation function can *kill* neurons by pushing them too far in the negative region during the backpropagation operation. Specifically, this means that the neuron's weight implies that it will always return an activation value of 0, which in turn yields a null gradient, making the unit unresponsive for the rest of the training process. Mathematically, we have:

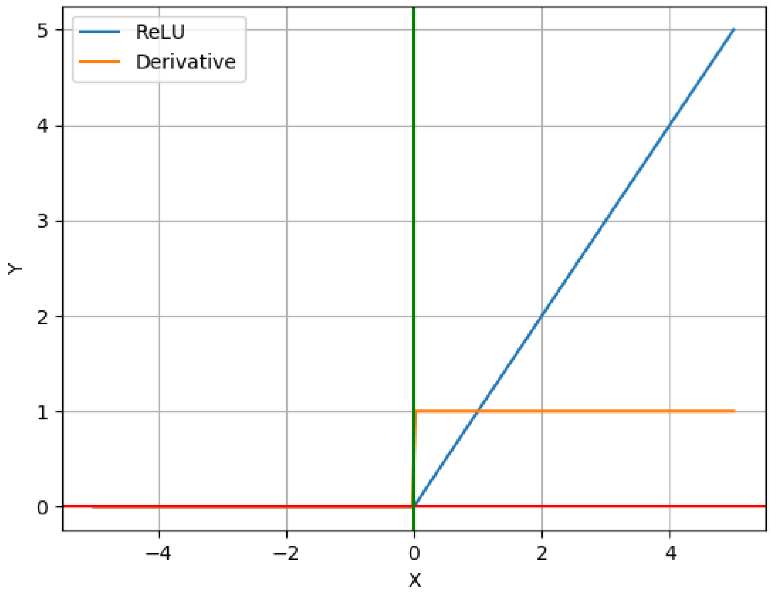


FIG. 2.6. The ReLU function and its derivative.

$$g_{\text{ReLU}} = \max(0, x) \tag{2.9}$$

2.2.3 Loss function and backpropagation for optimization

Deep learning models are usually constructed and trained to perform as well as possible in a given specific task. For a single task, their performance $J(W)$ depends entirely on their vector of parameters W. Consequently, two models with same architecture will exhibit different accuracies if their parameters are distinct. As such, the training process of a neural network means finding a set of parameters W that will reduce the model error that is usually computed through a loss function $\mathscr{L}$. While multiple training methods are still currently explored by the community, we focus here on the stochastic gradient descent, one of the most popular training procedure.

In a supervised learning context, and for a given set of input x and the expected vector y, the model m predicts an output vector $\hat{y} = m(x)$. The error can be computed by the evaluation of the distance between the model prediction and the expected value $e = \mathscr{L}(y, \hat{y})$. The model parameters w are then affected by the error gradients resulting from the cost function. The backpropagation algorithm allows to affect to each unit a gradient correction proportional to its participation in the average loss (39; 8). A common error function is the mean square error. In this case:

$$E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (y - \hat{y})^2 \tag{2.10}$$

where y is the target value and $\hat{y}$ is the neural network prediction. As explained above, this error should be minimized with respect to the network's weights. For the N examples, this can be expressed as:

$$\frac{\partial E(X,\theta)}{\partial w_{i,j}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial}{\partial w_{i,j}^k}\left(\frac{1}{2}(y_d - \hat{y_d})^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{i,j}^k} \tag{2.11}$$

Using the chain rule, it is possible to write the error partial derivative:

$$\frac{\partial E}{\partial w_{i,j}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{i,j}^k} \tag{2.12}$$

where a_j^k is the activation value of neuron j in layer k, before the activation function, as detailled in Equation 2.1. The first term in Equation 2.12 called the error, is denoted:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} \tag{2.13}$$

Using the activation value calculation, the second term in Equation 2.12 can be expressed as:

$$\frac{\partial a_j^k}{\partial w_{i,j}^k} = \frac{\partial}{\partial w_{i,j}^k}\left(\sum_{l=0}^{r_{k-1}} w_{l,j}^k o_l^{k-1}\right) = o^{k-1} \tag{2.14}$$

Consequently, the partial derivative of the error function E with respect to a weight $w_{i,j}^k$ is the product between the error term of neuron j in layer k and the output of node i from the previous layer.

$$\frac{\partial E}{\partial w_{i,j}^k} = \delta_j^k o_i^{k-1} \tag{2.15}$$

The error term computation depends on the neuron position in the network and starts from the final layer m. It can be expressed as:

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - g_o(a_1^m))^2 \tag{2.16}$$

where $g_0(x)$ is the activation function for the output layer. With the chain rule, the partial derivative is:

$$\delta_1^m = (g_o(a_1^m) - y)g_o'(a_1^m) = (\hat{y} - y)g_o'(a_1^m) \tag{2.17}$$

Finally, the partial derivative of the error function E with respect to a weight in the final layer $w_{i,j}^m$ is:

$$\frac{\partial E}{\partial w_{i,j}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y)g_0'(a_1^m)o_i^{m-1} \tag{2.18}$$

For the hidden layers, it is necessary to rely on the chain rule again:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} \tag{2.19}$$

Using the error value from the next layer enables to rewrite the equation as:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_j^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k} \tag{2.20}$$

With the activation a_l^{k+1} being $a_l^{k+1} = \sum_{j=1}^{r^k} w_{j,l}^{k+1} g\left(a_j^k\right)$, the second term in Equation 2.20 can be put as:

$$\delta_j^k = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{j,l}^{k+1} g'(a_j^k) = g'(a_j^k) \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{j,l}^{k+1} \tag{2.21}$$

Finally, by putting it all these elements together, the partial derivative error E with respect to a weight located in the hidden layers can be expressed as:

$$\frac{\partial E}{\partial w_{i,j}^k} = \delta_j^k o_i^{k-1} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r^{k+1}} w_{j,l}^{k+1} \delta_l^{k+1} \tag{2.22}$$

2.3 Supervised Learning

Supervised learning is the most straightforward approach when using deep learning and neural networks. This technique has been applied to a very wide range of tasks, featuring spam detection, image classification, video frame interpolation, system identification and regression tasks. Despite requiring a considerable volume of examples to be successful, recent supervised learning algorithms have improved the state-of-the-art in numerous challenging tasks (39).

In practice, it requires a dataset that relates input and output pairs, generally denoted (X, Y) and consists in finding statistical relationship between these elements. Specifically, with x_i a given input example in the dataset, the neural network f predicts a label $\hat{y}_i = f(x_i)$ which is compared to the dataset label and returns an error for this example. Mathematically:

$$e_i = \mathscr{D}(\hat{y}_i, y_i) \tag{2.23}$$

where $\mathscr{D}$ is a function that evaluates the distance between the predicted label and the real target. This error is then averaged with other examples from the dataset, to compute the neural network loss and thus the optimization gradients.

2.4 Reinforcement Learning

Reinforcement Learning (RL) algorithms are a class of machine learning algorithms that adresses the problem of optimal decisions over time (70; 125; 131). Specifically, it features an agent tasked to select actions in an environment in order to maximize a sum of rewards along the time it interacts with its world. RL is a particularly well-suited way to tackle robotics challenges as it is able to propose successful control policies without having to analytically define a model of the dynamics, which often proves to be challenging.

2.4.1 Principle and main concepts

Reinforcement Learning is a deep learning branch that, instead of training a model on a fixed dataset, features an agent interacting with an environment, which has also its own dynamics. Furthermore, the environment provides the agent with observations and rewards. A simplified view of this process is displayed Figure 2.7.
The agent's goal is to maximize the cumulative reward, also called return. While doing so, the agent learns how to make decisions for objectives defined by the reward function. Formally, environments are Markov Decision Processes (MDP), which are a 5-tuple $< \mathscr{S}, \mathscr{A}, \mathscr{R}, \mathscr{P}, \rho_0 >$ with:

- $\mathscr{S}$ is the set of all valid states,
- $\mathscr{A}$ is the set of all valid actions,
- $R : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \to \mathbb{R}$ is the reward function
- $\mathscr{S} \times \mathscr{A} \to \mathscr{P}(s)$ is the transition probability function that estimates the probability to reach a certain state based on the previous state and the action: $P(s'|s,a)$, with s' being the next state, s, a the state and the action respectively.
- ρ_0 the start-state distribution

Furthermore, a MDP must satisfy the Markov property which states that a transition depends only on the most recent state and action, setting the influence of previous states to zero.

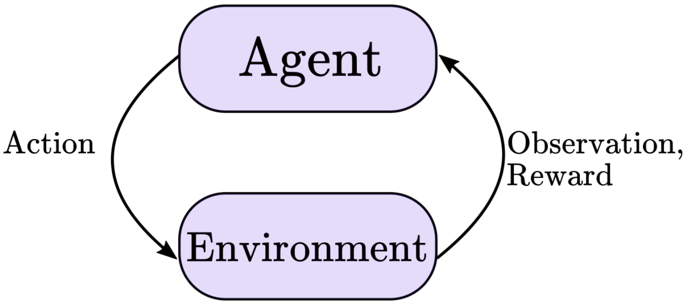


FIG. 2.7. RL principle.

2.4.1.1 Observations

The environment/world can be described by a state $s \in \mathscr{S}$. Depending on the environment considered, the observation $o \subset s \in \mathscr{S}$ that is provided to the agent is a subset of s, potentially omitting information. Most of the time, states and observations are real-valued vectors or higher-order tensors (RGB images, for instance). The reception of an observation precedes the agent's action selection.

2.4.1.2 Policy

The logic used by an agent to make decisions (that is, to select actions) is referred to as a policy. As an analogy, it can be seen as the agent's brain. In usual deep RL framework, these policies rely on parameters, such as the weights of the agent's neural network, often denoted as θ. They can either be deterministic, see Equation 2.24, or stochastic, see Equation 2.25.

$$a_t = \mu_\theta(s_t) \tag{2.24}$$

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{2.25}$$

Stochastic policies commonly belong to categorical policies or diagonal Gaussian policies, depending on the action space. Categorical policies are affiliated with discrete action spaces, while Gaussian policies are used whenever the environment features a continuous action space.

2.4.1.3 Actions

The state observation enables the agent to select an action *via* its current policy. The action chosen by the agent can potentially affect and alter the environment it interacts with. As different environment would provide different observations, they similarly allow different actions. As such, actions can be real-valued vectors, for continuous action spaces or indices, for discrete action spaces. In RL, entire classes of algorithms can only be applied to one setting or another, underlining the importance of the action-space description.

2.4.1.4 Trajectory and episodes

Most of the time, agents will alternate between state observation and action selection. As such, in the RL framework, a trajectory τ defines a sequence of observations or states provided by the environment and actions from the agent.

$$\tau = \{(s_0, a_0), (s_1, a_1), ..., (s_n, a_n)\}$$

The first state is sampled from the start-state distribution ρ_0:

$$s_0 = \rho_0(\cdot)$$

State transition depends both on the environment, which can have a stochastic part, and the most recent agent action, as required by MDP. Thus, for a deterministic transition, we have:

$$s_{t+1} = f(s_t, a_t)$$

while for a stochastic, one transition can be written:

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

In RL, an episode is a trajectory that ends with a termination state: a termination state is a state that calls for a reset of the environment. It can be the goal state (reaching a specific point in space, for instance) or an absorption state (the agent pushed the object it was supposed to grasp out of its reach). It is also possible to consider that an episode is over after a given number of steps.

2.4.1.5 Reward Function

The action selected by the agent after state observation is usually met by an evaluation from the environment: the reward. The reward function R holds a paramount importance in RL, as it is the main tool for shaping the agent behaviour. Formally, the reward function is a real-valued function $\mathscr{R}$:$\mathscr{S} \times \mathscr{A} \times \mathscr{S} \rightarrow \mathbb{R}$ that evaluates an action taken in a given state and its effect. That is:

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

Reward functions revet various forms. Specifically, two main classes of rewards can be referenced:

- Sparse rewards: Also known as impulse rewards, this class of functions affects a reward r_g to a small area of the explorable space, while most of the environment returns a reward r_l with $r_g >> r_l$. A classic example of sparse rewards is the Atari Pong environment (81) where a positive reward is only given to the agent when it scores, and inversely a negative reward is perceived when the agent lets the ball go through. Otherwise, the reward is null

- Shaped rewards: As opposed to a sparse reward, a shaped reward provides a learning signal in the majority of the explorable space, offering a gradient of reward to the agent that may sometimes ease the learning phase. A reward inversely proportional to the distance with the target point in a robotic reaching task is a common usage of shaped rewards.

Reward functions are not necessarily linear nor continue, thus giving an important flexibility to shape the agent behaviour. The sum of rewards over an episode is called the return.

2.4.1.6 Reinforcement Learning Objective

In RL, the ultimate goal is to train a policy π that maximizes the expected return $J(\pi)$ in an environment, also known as the performance, which corresponds to the average cumulative reward over a certain number of episodes. The expected return can be written as:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{2.26}$$

where $P(\tau|\pi)$ is the probability of a $T-$steps trajectory τ under policy π and can be written as:

$$P(\tau|\pi) = \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{2.27}$$

Consequently, the optimal policy π^* being the one that maximizes $J(\pi)$, it can be put as:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.28}$$

2.4.1.7 Value functions

To train the agent policy and improve the global performance, most RL algorithms embed a value function (131; 120; 122). Value functions are used to represent the desirability of a state. There exist several main value functions:

On-policy Value Function: $V^{\pi}(s)$, in this case, the value corresponds to the expected return starting from state s and using policy π until episode termination.

$$V^{\pi}(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{2.29}$$

On-policy Action-Value Function: $Q^{\pi}(s, a)$, also known as quality-function, represents the expected return starting from state s and taking action a and following the policy π afterwards:

$$Q^{\pi}(s, a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{2.30}$$

It is then possible to define the **optimal** value function and the **optimal** action value function which are similar to the previously defined functions but where the policy is the optimal policy π^*. Respectively, we have:

$$V^*(s) = \mathbb{E}_{\tau\sim\pi^*}[R(\tau)|s_0 = s] \tag{2.31}$$

$$Q^*(s, a) = \mathbb{E}_{\tau\sim\pi^*}[R(\tau)|s_0 = s, a_0 = a] \tag{2.32}$$

2.4.1.8 Bellman Equations

The four equations introduced in Section 2.4.1.7 are tied with a set of equations called the Bellman equations (5) that state that **the value of a starting state can be computed as the expected reward of this state plus the next state value**. Formally, this can be written for both the value function as well as the action-value function as:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi,s'\sim P}[r(s,a) + \gamma V^{\pi}(s')] \tag{2.33}$$

where the discount factor $\gamma \in [0,1]$ indicates weights the importance of future rewards. The closer the discount factor γ is to 1, the more significant future rewards are. (131)

$$Q^{\pi}(s,a) = \mathbb{E}_{s'\sim P}[r(s,a) + \gamma \mathbb{E}_{a'\sim\pi} Q^{\pi}(s',a')] \tag{2.34}$$

where $s' \sim P$ stands for $s' \sim P(\cdot|s,a)$ that indicates that the next state is sampled from the environment's transition probability. Similarly, $a \sim \pi$ and $a' \sim \pi$ mean that the action and next actions are sampled from the policy, based on the state $a \sim \pi(\cdot|s)$ or next state s', $a' \sim \pi(\cdot|s')$, respectively.
Bellman equations can be adapted for the optimal value functions as well:

$$V^{*}(s) = \max_{a} \mathbb{E}_{s'\sim P}[r(s,a) + \gamma V^{*}(s')] \tag{2.35}$$

$$Q^{*}(s,a) = \mathbb{E}_{s'\sim P}[r(s,a) + \gamma \max_{a'} Q^{*}(s',a')] \tag{2.36}$$

2.4.1.9 Advantage function

The advantage function is a way to discriminate useful actions. Mathematically, the advantage corresponds to the subtraction of the value function V from the action-value function Q.

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.37}$$

As can be deduced from this relation, an action better than the average would result in a positive advantage. This notion is at the center of various recent RL algorithms (121)

2.4.2 Practical aspects of RL

2.4.2.1 The Exploration-Exploitation trade-off

Usually, a RL agent has initially no specific knowledge about the environment it must interact with. In practice, this is represented by randomly initialized parameters for a neural network or a uniform state for a tabular approach. The exploration of the environment, through a trial-and-error process, is a way for the agent to gather knowledge and understand how to reach highly valuable states. Indeed, the more the agent knows about the environment, the quicker it can optimize the policy reflecting its behaviour. However, due to the potentially infinite size of a wide range of environments, pure exploration can not be considered as an efficient resources usage and is likely to confuse the agent. Similarly, systematic exploitation of available knowledge (selecting the current supposed best action) may result in sub-optimal or poor performance overall. Consequently, a successful RL algorithm must take into account these notions and find a trade-off between the two attitudes and an important quantity of works was devoted to this critical issue (4; 26; 103; 131)

2.4.2.2 Credit assignment problem

The credit-assignment problem is a frequently encountered issue in environment with sparse rewards. Specifically, as the agent seldom receives a reward signal, it is most of the time at the end of a trajectory. Consequently, a full sequence of states and actions was involved in order to get to the state that provided the reward. In these cases, it becomes difficult to effectively discriminate between useful and detrimental actions (81; 121). There exist numerous examples of such environments, such as the games of Chess or Go, or simulated environments like Pong, where the reward function sends a signal only on episode termination. In those, as the episode trajectories can frequently exceed several dozens of transitions, the credit assignment problem becomes obvious.

2.4.2.3 Sample (In)efficiency

While RL algorithms have drastically improved recently, training an efficient policy in the RL framework still requires a very large quantity of trajectories. In the widely recognized DQN paper (82), the trained agent never reaches 50% of human performance even after 200 million frames, which corresponds to over 900 hours of practice in real-time. More recently, the impressive results of OpenAI and in the Hide-and-Seek environment were obtained after several billions of frames. However, for simpler environments, it is still possible to train acceptable policies within an reasonable time frame. Along with potential cost of damaged robots during training, it is one of the main motivations for simulated environments.

2.4.3 Modern RL algorithms

This section introduces practical implementations of recent RL algorithms. Starting from the broad perspective of model-free or model-based approaches, it progressively narrows its focus until introducing the main RL algorithm used in this work: Proximal Policy Optimization, see Figure 2.8.

2.4.3.1 Model-free and model-based RL

The question of using model-free or model-based algorithms can be formulated as: **Does the agent have access to the environment model ?** This has heavy implications as having access to the environment model implies that the agent is aware of the functions governing the states transitions as well as the rewards. This hypothesis has practical upsides, as it allows the agent to plan its actions ahead, exploring various choices and weighting their consequences. A recent example of successful application of model-based techniques is AlphaGo(126; 127). However, in practice, a reliable environment model is seldom available, thus hindering the performances of the agent when used in the real

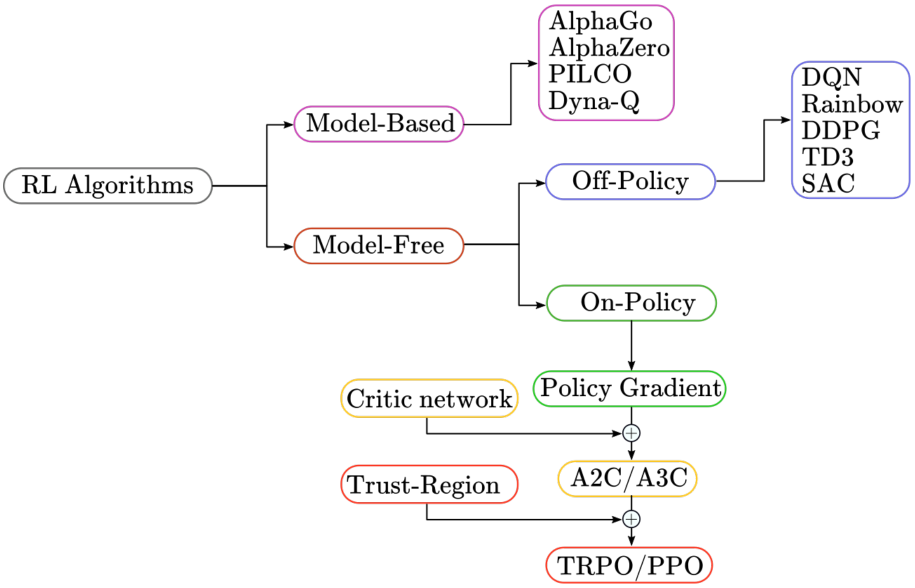


FIG. 2.8. A taxonomy of RL algorithms.

settings. An alternative approach is to learn the environment dynamics. In this configuration, agents usually alternate between learning the environment model and improving their policy. Nevertheless, this is likely to result in a biased model that is prone to be exploited by the agent. Although it can lead to satisfying results in simulated environments, it is often correlated with poor behavior once deployed to the real target environments. Techniques that renounce in using the environment dynamics are classified as model-free methods. They are currently very popular and widely used in the literature and, as the rest of this report relies on this class of methods, model-based will not be detailed further.

2.4.3.2 On-Policy vs. Off-Policy

Among model-free algorithms, another crucial branching divides again the algorithmic pool into two notable groups: on-policy or off-policy algorithms. On-policy algorithms optimize the parameters θ of a policy π_θ by performing gradient ascent on the gradient of their performance $J(\pi_\theta)$ or an approximation of it. Each update (optimization) relies on data collected by the latest policy version, which partially explains the sample inefficiency of these techniques. On the contrary, off-policy methods learn an approximator of action-values $Q_\theta(s, a)$ and their updates rely on the Bellman equation. Consequently, they do not require that the data provided for the update be generated by the latest policy. This presents various advantages, such as having the possibility to insert expert trajectories or human examples within the data used to train the policy. While this idea is appealing, as is the fact that off-policy algorithms are more sample efficient, they are also more brittle as they do not optimize directly for the performance (given that they optimize for $Q_\theta(s, a)$). In contrast, on-policy methods are more stable and reliable (as they directly optimize for the performance $J(\pi_\theta)$). The main RL algorithm used in this report is PPO and is an on-policy method.

2.4.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an on-policy, actor-critic with trust-region RL algorithm (120) introduced in 2017 (122). Despite its relative simplicity, this method is robust, scalable and reliable and has been at the center of various impressive RL achievements in recent works (100).

On-policy updates As explained in section 2.4.3.2, the goal is to maximize the agent performance that equates with maximizing the expected return:

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{2.38}$$

The gradient ascent update can be written:

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.39}$$

where $\nabla_\theta J(\pi_\theta)$ is the policy gradient, θ_k the policy parametrization at step k and α is the learning rate. To compute the policy gradient, an analytical gradient of the policy performance is necessary. It can be written as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)] \tag{2.40}$$

The updates resulting from Equation 2.40 increase the probability of an action rewarded positively, while discouraging the agent to select actions that were met with a negative reward.

Actor-Critic As can be seen in Equation 2.40, the probability to select an action is directly dependent on the sum of rewards obtained in the trajectory. In practice, however, this signal may be noisy and present an important variance. Actor-critic approaches were designed to mitigate the noise from the reward, leading to a faster and more robust training. Specifically, they rely on an additional network parametrized by weights ϕ, called the critic, that estimates the value $V_\phi(s)$ of each state. This scalar is then subtracted from the reward, allowing the action to be evaluated on whether or not it yields a superior return than the expected. Consequently, Equation 2.40 becomes:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau) - V_\phi(s_t))] \tag{2.41}$$

In these methods, the critic is also trained and thus provides only an approximation of the state value. The most common loss function for the value function is:

$$\phi_k = \arg\min_{\phi} \mathbb{E}_{s_t,R_t\sim\pi_k}[V_\phi(s_t) - R_t] \tag{2.42}$$

where the discounted return can be expressed as $R_t = \sum_{i=t}^{N} r_i\gamma^{i-t}$

Proximal Policy Optimization The PPO algorithm was designed to prevent a policy from taking too important update steps that harm the performance, sometimes beyond recovery, while still going as far as possible in the gradient ascent. To do so, it relies on the notion of trust-regions. Trust-regions were introduced in (120) and rely on gradient clipping in the objective function to avoid moving the new policy too far from the previous one. The PPO objective function can be written as:

$$L(s, a, \theta_{\text{old}}, \theta) = \min(r_t(\theta) A^{\pi_{\theta_{\text{old}}}}(s, a), \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) A^{\pi_{\theta_k}}(s, a)) \tag{2.43}$$

where $r_t(\theta)$ is the ratio of the probabilities between the previous policy and the current one for a given pair of state and action. This ratio can be expressed as:

$$\rho_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(s, a)} \tag{2.44}$$

and the *clip* function ensures that the ratio stays in the $[1 - \varepsilon, 1 + \varepsilon]$range

2.5 Generative Adversarial Learning

RL algorithms represent the go-to approach when considering control environments and are extensively used in the Universal Notice Network method presented in Chapter 4. Nevertheless, the unsupervised learning framework offers interesting concepts and some approaches can be repurposed in a context that is beyond their usual scope. This part introduces the Generative Adversarial Learning (GAN) (40) framework, an unsupervised generative modelling technique on which the CoachGAN transfer technique (86), one of our main contributions, see Section 5.1, relies heavily.

2.5.1 Principle

The concept of Generative Adversarial Networks (GAN) is rooted in Information Theory and generative modelling. In this paradigm, the goal is to estimate the underlying data probability P_{data} of a given dataset. As opposed to discriminative modelling which objective is to infer labels y from data samples, the optimization of a generative model yields a function able to generate unseen samples from a distribution matching the training dataset, as shown in Figure 2.9. Within the GAN framework, this process is implemented through a competitive setting involving two functions:

- a generator that creates samples
- a discriminator: an adversarial function that evaluates the samples

In this setting, the two networks share an objective function and are jointly trained using the following minimax game.

$$\min_\theta \max_\phi V(G_\theta, D_\phi) = \mathbb{E}_{x \sim P_{\text{data}}}[\log D_\phi(x)] + \mathbb{E}_{\theta z \sim P_z}[\log(1 - D_\phi(G_\theta(z)))] \tag{2.45}$$

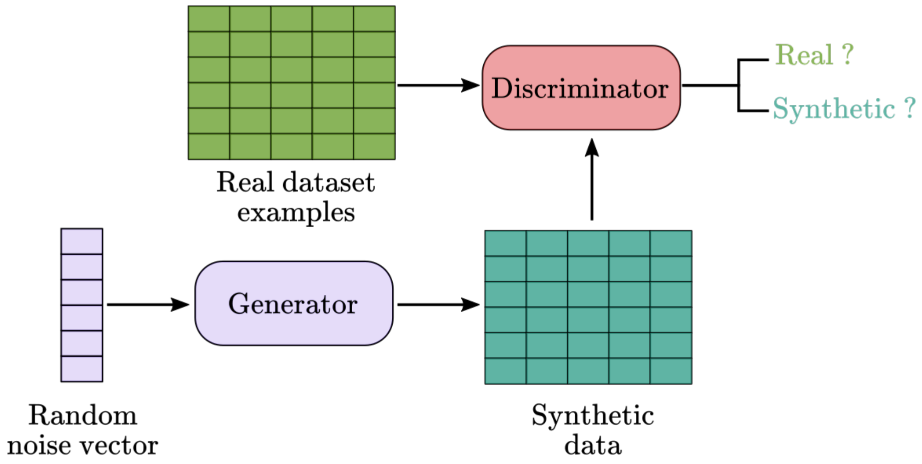


FIG. 2.9. GAN principle.

where G_θ is the generator, using a vector of parameters θ and D_ϕ is the discriminator relying on weights ϕ, P_{data} is the training dataset and P_z is a probability distribution for the latent variable z used by the generator to produce samples. This latent variable is, in practice, a noise vector sampled from a uniform or unit Gaussian distribution and then mapped to samples by the generator. In return, the discriminator D assigns a scalar value to the sample $G_\theta(z)$ representing how likely this sample belongs to the training distribution, thus acting as an adversarial classifier. Formally, Equation 2.45 states that the optimal discriminator must be able to perfectly distinguish between synthetic (generator-produced) samples and real samples from the training dataset. In practice means to respectively affect a value of $D_\phi(G_\theta(z)) = 0$ to synthetic samples and a value of $D_\phi(x) = 1$ to real samples. On the other hand, the optimal generator should be able to confuse the discriminator and produce synthetic samples of high likelihood with respect to the dataset that would systematically be classified as real by the discriminator. This adversarial setup consequently provides a suitable way to use each model to train the other one (i.e: the discriminator is penalized for each synthetic sample classified as real and vice-versa). The discriminator and generator loss function can respectively be expressed as:

$$\mathscr{L}_{\mathscr{D}_\phi} = \frac{1}{m}\sum_{i=1}^{m}[-\log(D_\phi(x^i)) - \log(1 - D_\phi(G_\theta(z^i)))] \tag{2.46}$$

$$\mathscr{L}_{\mathscr{G}_\theta} = \frac{1}{m}\sum_{i=1}^{m}\log(1 - D_\phi(G_\theta(z^i))) \tag{2.47}$$

2.5.2 Reaching optimality

As introduced in Equation 2.45, for a fixed generator G_θ, the discriminator does a binary classification. Practically, it should assign a value of 1 to data from the training set $x \sim P_{\text{data}}$ and a value of 0 to samples coming from the generator $x \sim P_{G_\theta}$. Thus, for a given sample x, the optimal discriminator can be written:

$$D_G^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + P_{G_\theta}(x)} \tag{2.48}$$

If the generator has perfectly captured the training distribution, then the optimal discriminator will assign a value of 0.5 to the samples, which equates to randomly guessing whether the sample is real or synthetic. On the other hand, optimizing the generator based on the optimal discriminator D_G^* evaluations is known as minimizing the Jensen-Shannon (JS) divergence between the training distribution and the synthetic distribution:

$$2D_{\text{JS}}(P_{\text{data}}||P_G) - \log(2) \tag{2.49}$$

While theoretically sound, this approach is debatable in practice due to the models architecture. Specifically, as the discriminator outputs probabilities (as its last activation function is a sigmoid), a highly confident discriminator results in small gradients that fail to give a proper improvement signal to the generator. Consequently, the adverserial pair is commonly trained jointly with the optimization process focusing alternatively on generator and discriminator. Several recent works were dedicated to these issues and propose alternative architectures that alleviate this problem.

2.5.3 Common issues and failure cases

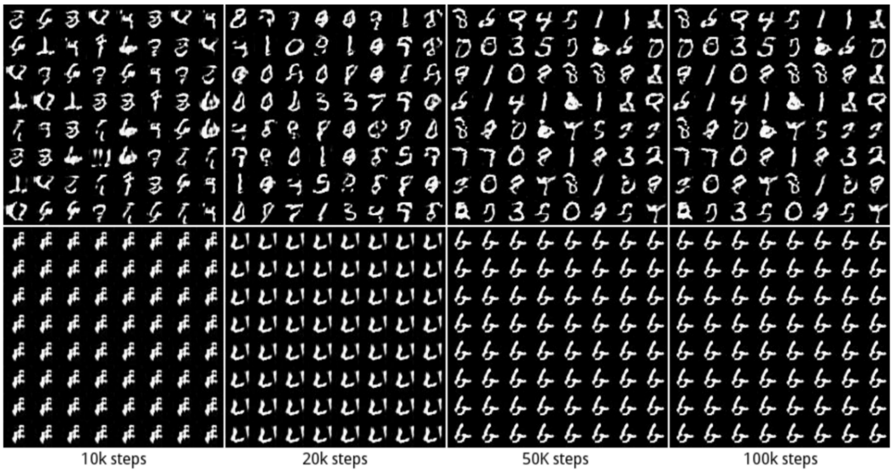


FIG. 2.10. Mode collapse during GAN training (78). The first row shows samples of a rather successful generator as the quality gradually increase as well as the diversity (each MNIST class is represented). The lower row displays an example of Mode Collapse where the generator focuses on a single point in the target space even though various noise vectors are provided.

Due to the adversarial part and the minimax aspect of the loss function, training GANs is notoriously difficult. While potentially affected by the usual issues of neural networks, generative adversarial networks are additionally subject to specific problems. Mainly :

- Unstability: As the existence of an equilibrium is not guaranteed, it is possible that the discriminator and the generator start a cycle of periodic domination. A symptom of this can be a sinus-like loss for both models and regular drop in generator samples. Carefully tuned hyper-parameters and model architectures can mitigate this issue.
- Mode collapse: In these cases, the generator fails to diversify its samples, basically mapping all latent vectors to a single point or a small area of the output space, which is considered real by the discriminator. Such a phenomenon is represented in Figure 2.10.
- Discriminator domination: In some cases, the discriminator initialization allows it to detect most of synthetic samples early in training, resulting in an increase in discriminator confidence that diminishes the gradients available for the generator.
- Overfitting: The generator is only able to generate samples that too much resemble the training set.

Despite its extended representative and modelling power, the GAN framework offers little straightforward process to control the latent space. Various works have been able to identify the main latent space axes by examining large quantities of samples and determining the transformation vector between two elements. Alternative modelling approaches, such as the Variational AutoEncoder, the main concept of the Task-Specific Loss (TSL), see Section 5.2, constraints the latent space to be more manageable, which in turn endows the user with a closer supervision on the model output.

2.6 Knowledge representation

For a large class of problems, finding the good set of features to represent the datapoint constitutes in itself an important part of the solution. However, it is most of the time unclear how to find these features. For instance, let us consider the problem of finding a human face in a picture. One could consider that a face has been found if a nose and two eyes can be detected. Nevertheless, it is not straightforward to provide the description of such a feature in pixel space, as eyes and nose can have various shapes, size, and color. Although it is manageable to define space boundaries for these elements, it is also necessary to deal with various lighting setting, shadows and element occlusion (due to face orientation, for example). A common approach in modern machine learning is to assign the feature representation to the model as well as the discovery of the relation between the input and output. Among the various architectures that implement this concept, the AutoEncoder (AE) is the most notorious instance (39; 8).

2.6.1 Autoencoders

Autoencoders are neural networks that are trained to learn to copy the input they are presented. While this may seem like a trivial task that could be accomplished with an identity function, autoencoders present a more sophisticated architecture that gives them interesting properties. Specifically, autoencoders can be decomposed into two functions with well established purposes:

- an encoder f_θ, parametrized by weights θ, that projects the input into a small abstract representation, also called the **code** $h = f_\theta(x)$
- a decoder g_ϕ that reconstructs the **code** in the initial space $\hat{x} = g_\phi(h)$

A usual autoencoder architecture is shown in Figure 2.11.

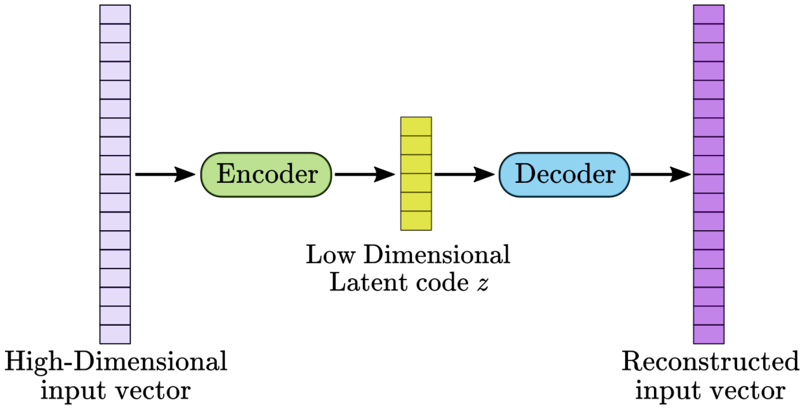


FIG. 2.11. An autoencoder architecture.

As mentioned, if the model were to learn an identity function, its usefulness would be questionnable. Consequently, the **code** created by the encoder is usually a vector which dimensionality is importantly lower than the initial input. Thus, the decoder must retrieve the initial signal with compressed information. This forces the whole model to learn which features should be prioritized. Consequently, the **code** produced by the encoder captures the most salient parameters of the training set.

The objective function for this class of models aims at minimizing the distance between the initial input and its reconstructed version. Formally:

$$L(\theta, \phi) = ||x - \hat{x}|| = ||x - g_\phi(f_\theta(x))|| \tag{2.50}$$

While the straightforward *vanilla* autoencoder approach may appear to have limited use cases, despite having the properties of a non-linear PCA (Principal Component Analysis), several recent works have applied AE variants to a large set of cases, demonstrating the architecture usefulness. For example, the denoising AE proposes to replace the initial input x with corrupted data and train the model to reconstruct the noise-free version:

$$L_{\text{DAE}} = ||x - g_\phi(f_\theta(x))|| \tag{2.51}$$

where $x_n = x + n \sim \mathcal{N}(\mu, \sigma)$ is the corrupted version of the input.

Other more recent works leverage the **code** representational capacity to repurpose AE architecture for generative modelling

2.6.2 VAE

Variational Autoencoders (62; 39)are an AutoEncoder alternative designed for generative modelling. In practice, this approach presents a crucial modification with the *vanilla* AutoEncoder. Namely, they are designed so that the constructed latent space is continuous, allowing samples drawn from the code distribution to produce realistic outputs once decoded.

In practice, the AE architecture is also modified to transform the previously deterministic encoder into a stochastic version. Specifically, the last encoder layer that was ultimately projecting the processed inputs to the latent space is replaced by a two-headed layer that outputs instead a vector of means μ and a vector of standard deviations σ, shown in Figure 2.12.

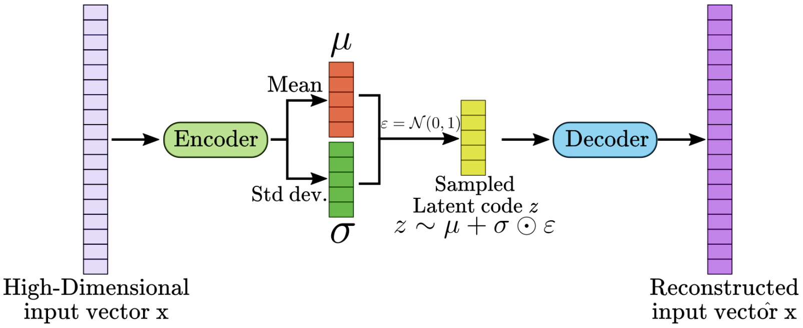


FIG. 2.12. From AE to VAE: a probabilistic encoder projects the input in a Gaussian constrained latent space.

Consequently, the **code** is not the result of a series of deterministic operations but is rather sampled from a normal distribution governed by the parameters generated by the encoder. This means that even for a same input, the **code** is likely to be slightly different for each pass, resulting in a probable area instead of a single point for the reconstruction. As a result, this effectively trains the decoder to be less sensitive and generalizes to latent areas instead of specific encodings as it is exposed to more variations. However, relying on the reconstruction loss alone with the sampling mechanism is not enough to create an interpolable latent space. Indeed, as there is no limit nor constraint on the mean vector μ and the standard deviation vector σ, the encoder is likely to cluster the different classes, generating distribution parameters that are very far in the latent space, thus creating a sparse latent space.

To prevent this, the loss function for VAE introduces an additional term that aims at regularizing the probability distribution of the **code**:

$$L(\theta, \phi) = -\mathbb{E}_{z \sim f_\theta(z|x)}[\log g_\phi(x|z)] - \mathbb{KL}(f_\theta(z|x)||p(z)) \tag{2.52}$$

In the expression in Equation 2.52, the first term is the reconstruction loss. It is the expectation of the negative log-likelihood of a datapoint x. This value depends on the encoder distribution over the representation $f_\theta(z|x)$. The regularization on the encoder distribution is enforced by the second term. It introduces the Kullback–Leibler divergence (KL divergence) (64) within the objective function. The KL function computes the divergence between two probability distributions. Hence, reducing the KL term means ensuring that the encoder outputs **codes** that are likely under the target distribution. In practice, the target distribution is usually the unit-centered Gaussian distribution $\mathcal{N}(0,1)$ because it offers a tractable expression for the KL divergence as well as an easy way to sample once training is over. An example of code distribution on the classic MNIST handwritten digits dataset using the various losses involved in VAE training is shown in Figure 2.13.

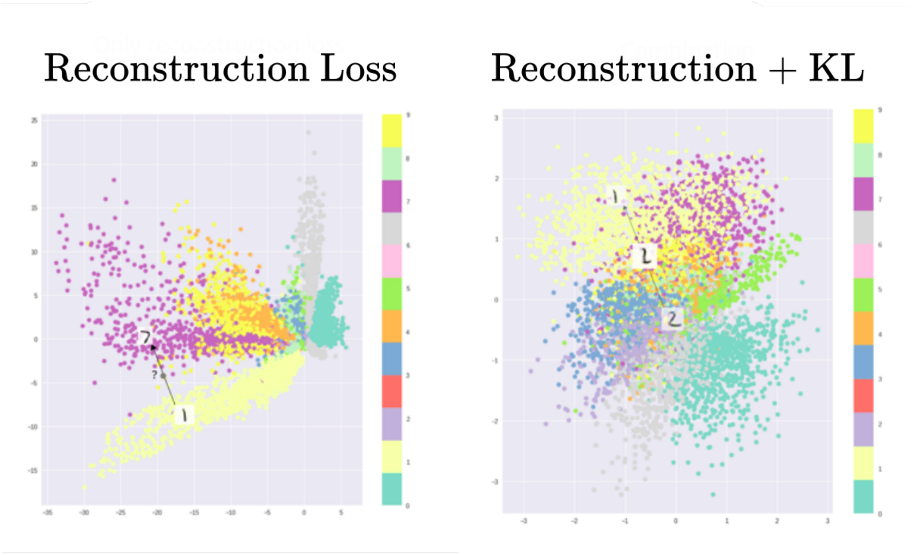


FIG. 2.13. Code distribution for VAE on the MNIST dataset.

Despite their simplicity, VAE have recently been in the center of a large spectrum of generative modelling works, spanning from images, synthetic text or even music. They are generally more reliable than their GAN counterparts, but they tend to produce blurrier output, due to the sampling mechanism trade-off.

2.7 Conclusion

This chapter introduced the major concepts that support the tools and frameworks developed in the manuscript. Going beyond simple feed forward neural networks, it presented the main ideas behind modern Reinforcement Learning (RL), which are used extensively in TAsk-Centered approaches presented in this report. To provide the necessary materials to fully understand TEacher-Centered techniques, several important unsupervised learning processes were also introduced such as Generative Adversarial Networks (GAN) and Variational AutoEncoder (VAE).

Chapter 3

Transfer Learning & Related works

Contents

3.1 Societal Needs for Transfer Learning

Industry 4.0, as foreseen in (123), will correspond to an era of higher integration of cyber and physical systems. In this views, industrial actors expect a strong interconnection between machines and devices, comprehensive information from the technology implemented and the ability of cyber systems to take decentralized decisions. These design principles are driven by the need for increased flexibility in terms of tasks, short industrial product series, consequently easily reconfigurable workplaces and finally highly diverse pool of robots.

This very dynamic industrial context introduces new and complex challenges that may prove difficult to formalize in mathematical frameworks. Indeed, perform complex motions to complete a task is still an open issue. While there exist optimization methods based on complete models (59; 60; 67), these are usually computationally intensive and may not satisfy real-time constraints. To tackle this issue, the community has also been leveraging predictive methods that are completely suitable for real-time control. However, this execution efficiency is usually achieved by sacrificing the model complexity and thus considering simplified versions that may not have enough degrees of freedom to express the full problem spectrum (22). These two methods present the strong and undeniable advantage of allowing direct transfer of skills between different agents. However, while it is possible to conceive hybrid controllers that can mitigate each approach's weaknesses, they nevertheless rely on an analytical model for both the task and the robot that can be hard if not impossible to define. As a result, it is common to integrate various hypotheses and assumptions that highly impact the precision of the model.

Recently, data-driven approaches, in particular deep learning based approaches, have been on the rise, as they can be used to find a nearly optimal solution in order to speed up the computation time in execution, while still being able to represent highly nonlinear processes. As the amount of data recorded and generated in the I4.0 processes is considerable, there appear to be strong incentives for learning-based approaches to become one of the go-to frameworks in the future.

As such, this chapter introduces the main principles and ideas on which Transfer Learning relies and proposes an overview of its applications in current literature.

3.2 Transfer Learning Principles

Transfer Learning, also sometimes referred to as domain adaptation, corresponds to configurations where it is desirable to exploit elements or *features*, that were learned in a given domain to improve generalization in another. Transfer Learning has become a most valuable tool in modern deep learning due to the size of recent architectures, in particular in domains such as Computer Vision (CV) and Natural Language Processing (NLP). Specifically, in such situations, the model considered must perform several tasks. In order for the transfer to be relevant, it is usually assumed that the distribution of tasks share multiple factors that can be captured by the model and increase the model performance on a task T_2 after having been trained on a dataset from T_1. As an example, a model trained on a dataset containing a large quantity of images, such as the commonly used ImageNet, is likely to develop pattern detectors that would be useful in another dataset. Practically, it the low-level convolutional kernels learned by a CNN are often reused as-is, only changing the latest model layers to fit the new task label space. This process is depicted in Figure 3.1.

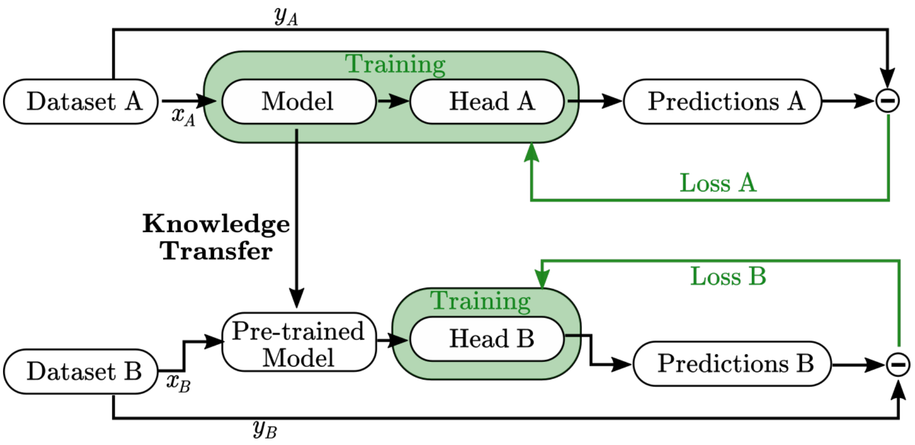


FIG. 3.1. Transfer Learning principle.

Formally, following works from (93), we define a domain $\mathscr{D}$ as a two-element tuple consisting of a feature space $\mathscr{X}$ and a marginal probability $P(X)$ where $X \in \mathscr{X}$. A task can also be defined as a two-element tuple: the label space $\mathscr{Y}$ and a conditional probability distribution $P(Y|X)$ that can be learned from the dataset. Given these conventions, and with $\mathscr{D}_S, \mathscr{D}_T$ and $\mathscr{T}_S, \mathscr{T}_T$, a source domain, a target domain and their respective tasks, it is possible to define Transfer Learning as the act of learning the conditional probability in the target domain $P(Y_T|X_T)$ with the insights gained from the source domain $\mathscr{D}_S$. Frequently, in practice, the target domain contains a significantly smaller number of labelled examples, thus giving its full importance to the initial training on the source domain. In this context, there are various scenarios foreseeable in Transfer Learning:

1. $\mathscr{X}_S \neq \mathscr{X}_T$: The feature spaces are different. In CV, this could for instance correspond to a source domain with RGB images and a target domain with grayscale data.

2. $P(X_S) \neq P(X_T)$: Difference in marginal probability distributions. This is one of the most common configuration where a pretrained model on a considerable dataset is retrained on a much smaller one.

3. $\mathscr{Y}_S \neq \mathscr{Y}_T$: Different label space. Closely linked with item 4

4. $P(Y_S|X_S) \neq P(Y_T|X_T)$. This case is also very frequent. One particular instance would be to reused a model trained on a classification task to do depth estimation.

Architecture Name	Year	Error Rate	Parameters (M)
AlexNet (63)	2012	16.4	60
VGG(128)	2014	7.3	138
GoogLeNet(136)	2015	6.7	4
Inception-V3 (134)	2015	3.5	23.6
Inception-V4 (133)	2016	4.01	35
Inception-ResNet (133)	2016	3.52	55.8
ResNet (48)	2016	3.6	25.6
ResNeXt (146)	2017	4.4	68.1

TABLE 3.1. Recent CNN architectures, performances on the ImageNet classification task and number of parameters (millions).

The following sections focus on modern usage of Transfer Learning and relevant work in this area. It begins by considering the most common cases, in Computer Vision (CV) and Natural Language Processing (NLP), that heavily benefits from this technique, due to the very important size of the models involved in these tasks. It then focuses on applications of this concept in control tasks. Specifically, this second part explores applications of this concept in Reinforcement Learning (RL), from the various process set up to transfer skills between tasks, to strategies for transfer from simulated environments to real physical robots. Finally, we also present a short overview on how unsupervised settings and knowledge representation can be useful to reach the transfer objective.

3.3 Common cases of Transfer Learning

As mentioned in the above section, neural network architectures have been investigated for more than 40 years, but it was the impressive accuracy improvements demonstrated during the 2012 ImageNet competition that boosted this approach popularity. Specifically, AlexNet (63), the winning architecture was based on Convolutional Neural Networks (CNN) (66), a partially-connected layer relying on convolution filters. Convolution filters, also known as kernels, offer several advantages over fully-connected architectures such as the possibility to capture spatial dependencies, a reduction to the overall number of parameters as well as the ability to learn image processing filters that would have to be otherwise manually-engineered such as edge or corner detection.

CNN are consequently appealing for Computer Vision (CV) tasks and have been applied and repurposed in an important quantity of tasks and architectures, regularly improving accuracy records. Table 3.1 (61) displays a short recap of the most recent and notable architectures in this domain. As can be seen, the decreasing error rate on the ImageNet classification task is often correlated with a considerable number of weights to train.

Although impressive by their size, the CV models presented in Table 3.1 appear thin and lightweight compared to recent NLP architectures. Indeed, the Attention mechanism introduced in (139) has led the NLP community to a new type of architecture, called the Transformer (139), that replaced the recurrent aspect of neural networks with attention

Architecture Name	Dataset size (GB)	Parameters(M)	GLUE score
GPT (104)	5	117	72.8
GPT-2 (105)	40	1542	80.1
BERT (21)	18	340	81.9
XLM (16)	18	665	82.8
RoBERTa (73)	18	665	88.5
ELECTRA (13)	116	340	88.6
XLNet(147)	116	340	90.5

TABLE 3.2. Recent NLP architectures characteristics: Number of parameters (millions), dataset training size and evaluation score on the GLUE benchmark.

layers. Their superior efficiency, computation-wise, ensured a wide adoption of this approach and a AlexNet moment for NLP. Since then, models have similarly grown in size and in capabilities, as presented in Table 3.2.

Due to the important capacity of these neural networks, training these models meets various challenges:

- Training requires considerable computational resources, hardly available on common hardware (training sessions can last up to several days on clusters of GPUs)
- Similarly, usage of these models is resources-greedy, making them unfit for real-time on most common devices
- The large model capacity makes overfitting highly likely, which requires consequently massive datasets with a substantial diversity

To face these issues, the community responded with various approaches. These can imply architectural modifications to reduce the number of parameters, such as those that support the MobileNet (52; 115) architecture. Formally, it uses a dedicated CNN, known as depthwise separable convolutions, which enabled this model to reach a 70.6% accuracy on ImageNet task with 4.2M parameters in its initial version. Comparatively, GoogLeNet (136) and VGG16 (128) that were then the best models present an accuracy score of 69.8% and 71.5% and hold 6.8M and 138M parameters, respectively. These techniques present substantial improvements in terms of usability for CNN and can be combined to greater efficiency. An additional way to reduce the costs of training large CNN models is to rely on Transfer Learning. Transfer Learning takes advantages of pre-trained models to repurpose most of the filters for another different task.

According to the characteristics of modern architectures, Transfer Learning, also sometimes referred to as fine-tuning, is widely used in CV and NLP. This section presents notable settings for this process and explains the main supporting mechanics in both task domains. This understanding will thus provides the tools to better recognize the challenges associated with the transfer of knowledge between models.

3.4 Computer Vision

Classification

In CV, Transfer Learning has been initially considered for classification tasks. Indeed, models (that traditionally include VGG (128), ResNet (48), SqueezeNet (55), DenseNet (54), Inception (133), GoogLeNet (136), ShuffleNet (151), MobileNet (52; 115), ResNeXt (146), Wide ResNet (150)) trained on a reliable dataset, such as Imagenet (more than 10^7 images for 1000 classes), have learned useful features in their convolutional layers. The term useful feature can refer to a given layer activation on the forward network pass or directly the weights of the convolution filter. In practice, these can recall manually-crafted edge or pattern detectors. Figure 3.2 shows an example of such filters. Specifically, these filters extracted from the convolutional blocks of a VGG 16 model pretrained on ImageNet show that the deeper the convolutional block, the more complex the filters.

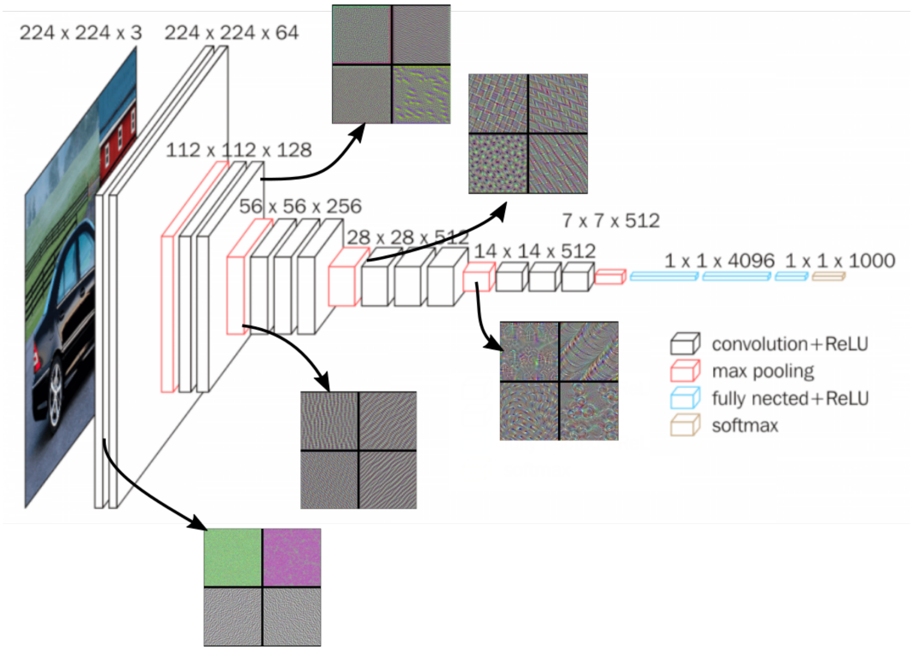


FIG. 3.2. CNN filters detect different features based on their depth location in the model.

To leverage these learned features and transfer them (fine-tune the model) to another classification task, the most direct and straightforward way is to replace the last network fully-connected layers by another adequately shaped fully-connected layer. In this configuration, it is common to refer to the pretrained model as a **feature extractor**. The rearranged model is then trained on the new task without modifying the convolutional layers (the feature extractor is said *frozen*. During the transfer, the frozen convolutional layers can be progressively incorporated back into the trainable parameters if required by the performance. This process, used in (24; 106; 148) can be seen in Figure 3.3

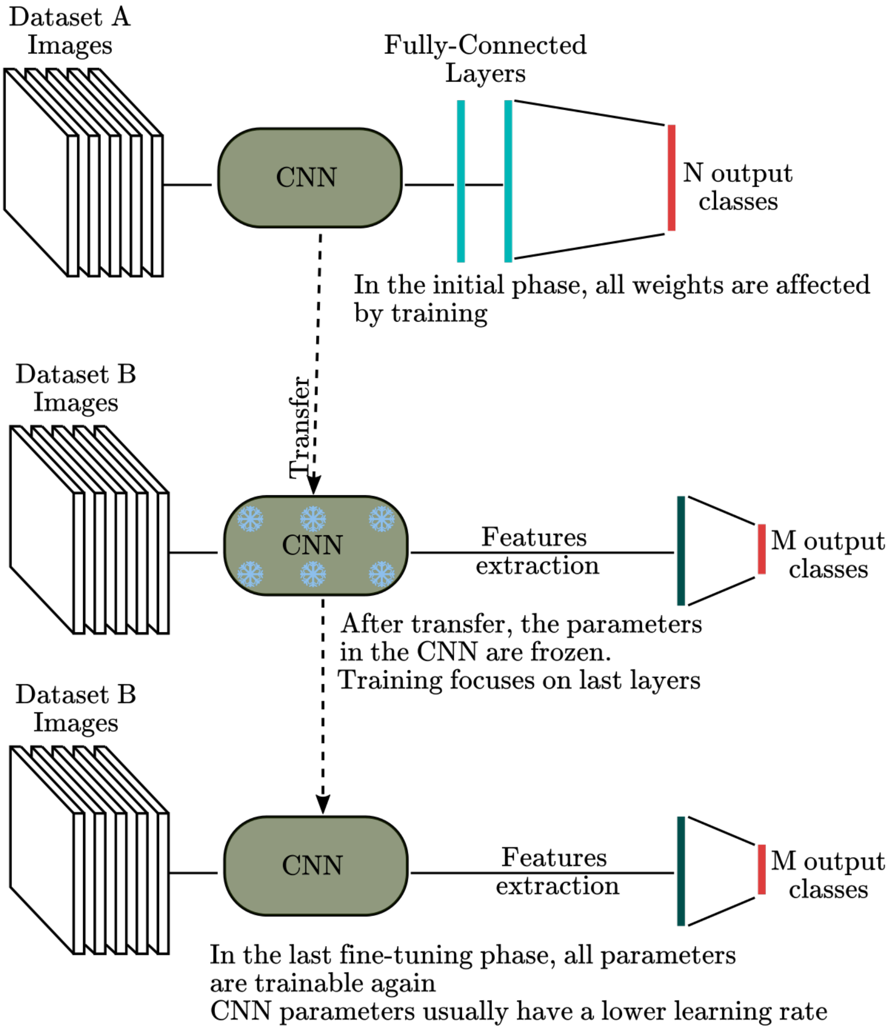


FIG. 3.3. CNN fine-tuning/Transfer Learning process.

Detection

Although fine-tuning/Transfer Learning is the most directly applied to classification tasks due to the fact that the initial model, sometimes refered to as backbone, was trained on such tasks, it is possible to repurpose the filters of a CNN to a wider spectrum of applications. Among these, it is usual to reuse a pre-trained CNN model to perform detection. R-CNN (37) and its subsequent versions Fast R-CNN (36) and Faster R-CNN (110) rely on a VGG backbone to propose regions of interest and associate them to the detected object class. Similarly, You Only Look Once (YOLO) (107; 108; 109) uses DarkNet-53 weights (although it is compatible with several backbones architecture) and is able to predict simultaneously bounding boxes and object classes. Figure 3.4 proposes a schematic view of this backbone integration along with the result of a forward pass in a detection architecture.

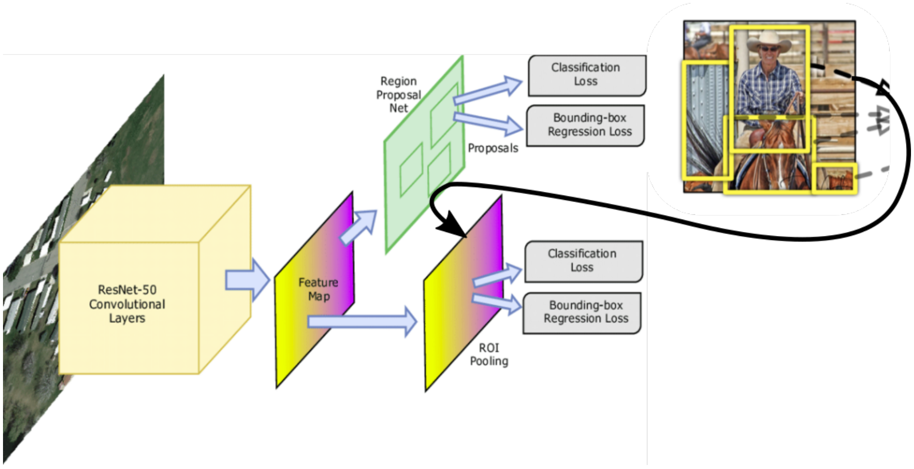


FIG. 3.4. Left: Detection model based on a pretrained backbone (37).

Segmentation

Image segmentation is a crucial topic in CV due to its prevalence in key areas such as scene understanding, autonomous driving or medical image analysis. Image segmentation nevertheless raises several challenges such as dense predictions. Indeed, as opposed to classification tasks or even detection for which models output a vector with a relatively restrained dimensionality, image segmentation techniques are expected to return image-shaped tensors, see Figure 3.5.

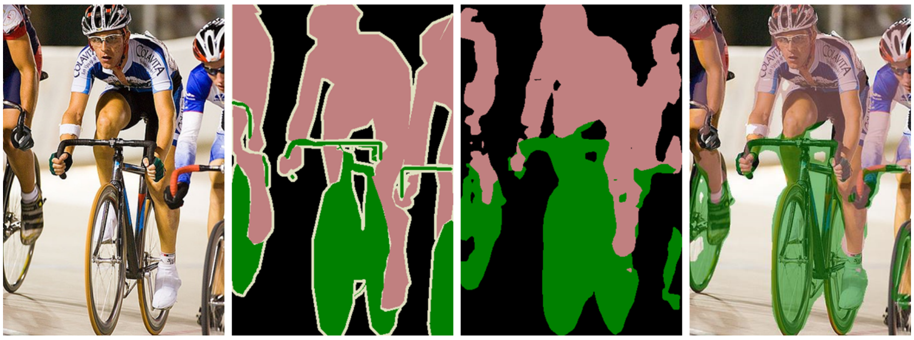


FIG. 3.5. Segmentation results (30).

Various approaches are proposed to meet this demanding expectations, among which:

- Fully Convolutional Networks(FCN) (124): FCN use only convolutional layers, thus allowing images of arbitrary size. Architecturally, a deconvolutional network follows a pretrained backbone (such as VGG16) and the last layer outputs class prediction for every pixel in the image

- Encoder-Decoder Based Models (112): These architectures also rely on a backbone, but are organized in a U or V shape that uses skip connections to reduce the loss of information between layers and increase image consistency.

Figure 3.6 shows various architecture models. As can be seen, both construction rely on a pretrained backbone and differ mostly in the way they connect this educated part with the new filters designed to complete the segmentation task.

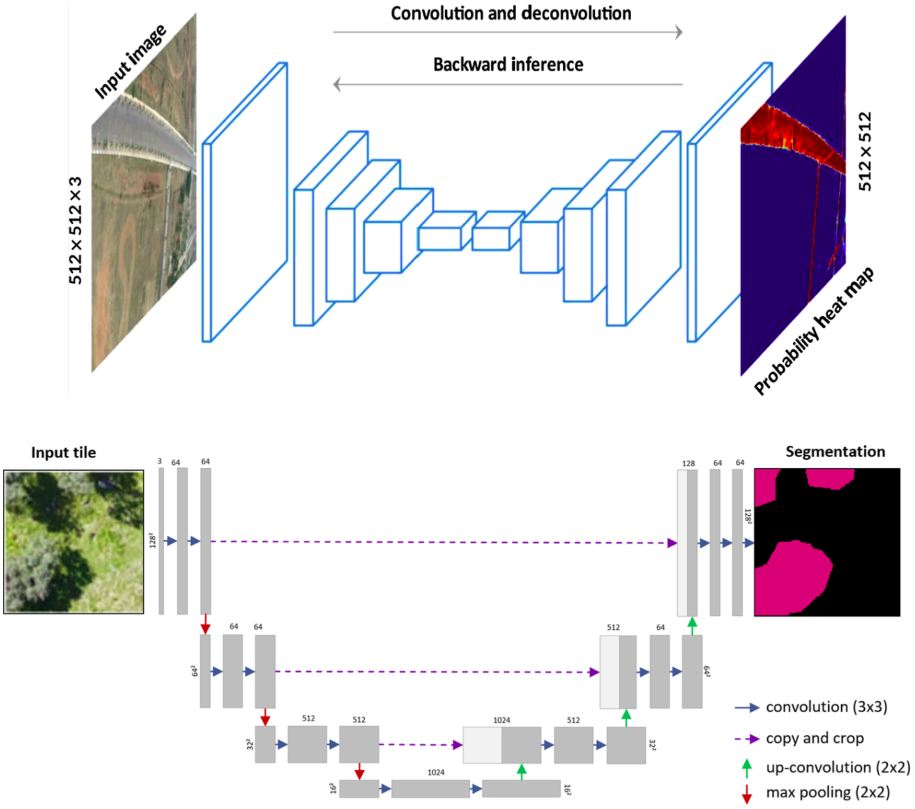


FIG. 3.6. Segmentation models. Up: Fully-Connected Networks (FCN) (124). Down: U-Nets (112).

As was demonstrated in this part, Transfer Learning is largely applied in deep learning-based CV. This process is heavily supported by the fact that training a consequent model on a large dataset results in the generation of useful detectors within the convolutional layers. Indeed, as these architectures present a strong dimensional bottleneck, the information is mostly located outside of the fully connected layer. The convolutional layers can then be transferred and repurposed for a wide range of tasks as was shown in recent works. Let us now consider the application of Transfer Learning in NLP.

3.5 Natural Language Processing

Transfer Learning is a particularly common technique in the field of NLP. From **Word2Vec** to current language models, Transfer Learning has been correlated with state of the art in NLP in a vast range of tasks. This section focuses on Sequential Transfer Learning, a process similar to those described above in Section 3.4, where the target task is considered after the initial model is ready.

Word representation

As neural networks are only able to process vectors, the first step in a training process is consequently to convert the text corpuses into elements understandable by the model. It is for instance possible to create a vector representation of each unique word in the corpus. As such, **Word2Vec** groups the vector of similar words in the representation space. This model is trained through an objective function that expect the model to assign probabilities to the context (n words before and after) of a target word. This training approach is called Skip-Gram (80). The resulting representation, called word embedding, presents useful properties that go beyond the clustering of similar things and ideas. Indeed, the geometry in the representation space shows that the distance between the words *Rome* and *Italy* is similar to the distance between *Beijing* and *China*, see Figure 3.7.

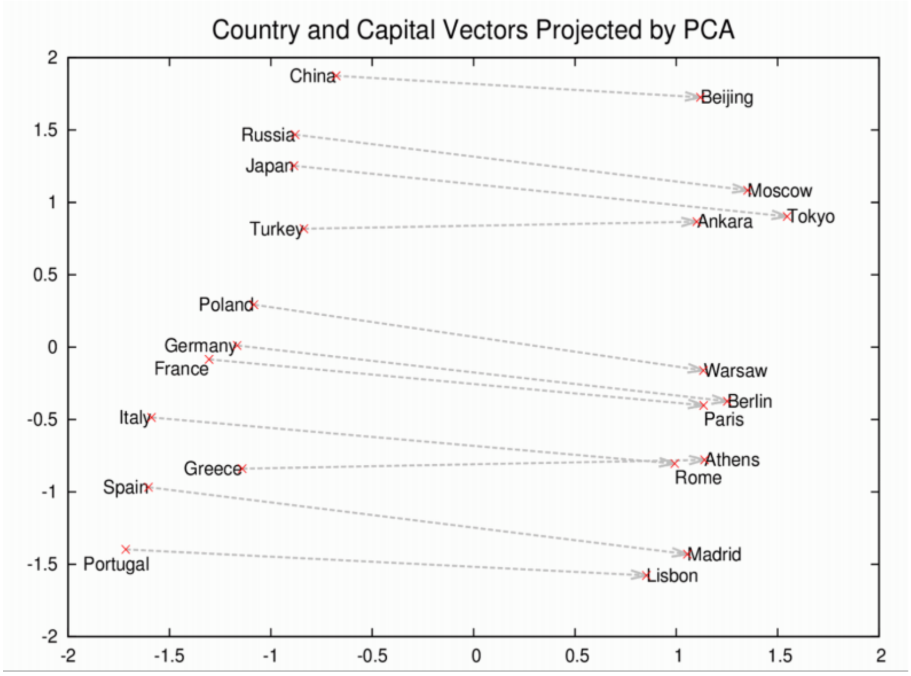


FIG. 3.7. PCA representation of words (79).

It is consequently easy to understand how these learned representations can be useful for a language model that could rely on the word embedding for a text-related task.

Language Models

The increasing popularity of the Attention mechanism (139) in neural-network has given an important traction to the Transformer architecture, which represents now the go-to method for text-related tasks. In particular, BERT and all its subsequent variants (RoBERT (73), DistillBERT (137) showed state-of-the-art performances on GLUE, SQuAD v1.1 among other tasks. Language Models (LM) are trained on immense corpuses on a simple objective: next word prediction based on all of the previous words within some text

(53; 105). Additional tasks can be simultaneously considered such as classifying whether two sequences are located next to each other in the text ((21; 65; 77)).
This initial training phase, usually referred to as *pre-training*, aims at endowing the model with an understanding of a given language. Consequently, language models are by essence designed to be used in a Transfer Learning setting as a **feature extractor**. To do so, it is usual to apply the same strategy than in the CNN case. Specifically, remove the last fully-connected layer and add one or more linear layers on top of the pre-trained model, see Figure 3.8.

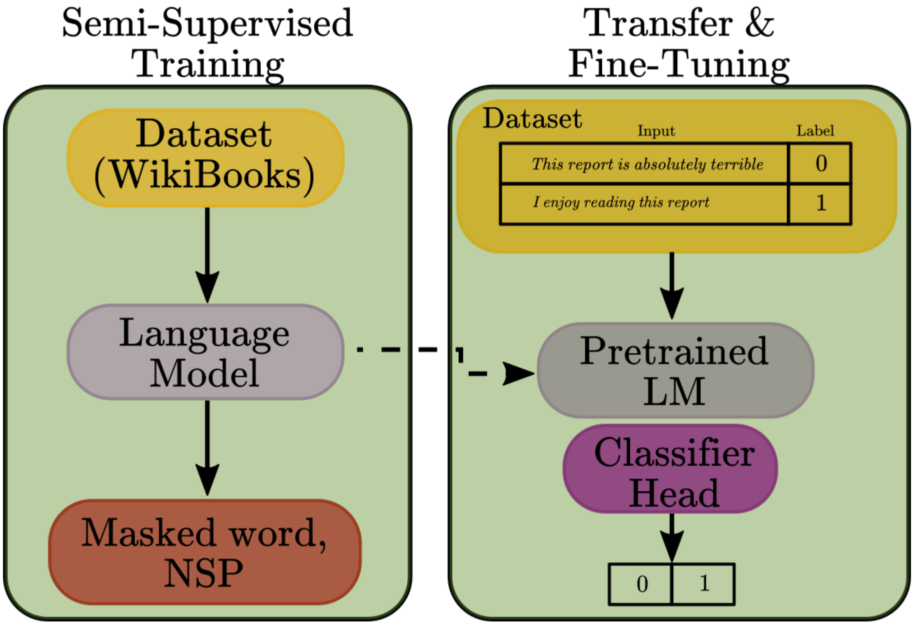


FIG. 3.8. Language Model based Transfer Learning.

3.6 Transfer Learning in control tasks

The previous section demonstrated the ubiquitous presence of Transfer Learning in CV and NLP settings. A common denominator of these domains is that all the associated sub-tasks share a single support (pixels in the case of CV and embedded words for NLP), thus yielding a straightforward way to reuse previously learned pattern detectors and feature extractors. Additionally, the dimensional bottleneck of common architectures in these domains naturally acts as a knowledge segmentation frontier. Such mechanisms do not have direct counterparts in the very diverse control environments and thus call for alternative strategies and paradigms when considering transfer in these configurations.

3.6.1 Deep learning based control

In modern data-based approaches, control tasks such as controlling a robot in a manipulation setting is usually adressed with RL techniques, rather than supervised learning.

Since the demonstration that it is possible for a RL agent to learn a human-level controller using only raw state observations (82), Deep Reinforcement Learning (DRL) has been increasingly popular. Recent algorithmic advances, such as Proximal Policy Optimization (PPO) (122) or Soft-Actor Critic (SAC) (44) have enabled the application of RL principle to complex environments, with non-linear dynamics and continuous action space, in simulation as well as in real-world (12; 20; 96; 132) conditions. For instance, the works described in (99) demonstrate that RL is a suitable way to control high-dimensional morphologies to accomplish accurate locomotion tasks, such as having a simulated T-Rex shaped character that dribbles a ball with its foot toward a moving target. These impressive results are nevertheless particularly expensive in terms of samples collections, limiting the direct application of such methods to real-world settings. Efforts are however deployed to tackle this issue. As such, in (44) the authors present the SAC algorithm, an off-policy RL agent, and prove that not only this approach is very efficient for the usual pool of simulated tasks, but also suitable for a physical task using for instance a Minotaur robot. The agent is trained from scratch, without simulations nor demonstrations, to maximize its forward velocity. This challenging task, although brilliantly managed by the agent, is nonetheless hardly reproducible for more complex tasks. Indeed, samples collections in the Minotaur setting are relatively unexpansive and fast, while it is likely that tasks of enhanced sophistication may require more costly samples. Consequently, the challenges yielded by this situation invoke one of the main transfer justification for control tasks: transferring knowledge acquired in simulation to real-world settings.

3.6.2 From simulation to real world

As hinted above, despite its recent achievements, DRL is notoriously sample inefficient (7; 29; 41; 71; 81; 88; 95; 101). On-policy algorithms in particular require millions of interactions with an environment to learn an acceptable policy. While training in simulation offers notable advantages, such as parallelizable policies, inexpensive samples collections, respect of safety concerns for operators as well as for the agent's body, simulated settings seldom offer a reliable copy of physical devices, which results in brittle and inefficient policies if directly deployed to a real-world setting, as the target domain differs from the training configuration. A possible option to mitigate these difficulties is to modify continuously the simulated environment to lessen the agent sensitivity to environmental changes and focus on the important features of the current task. In (97; 102), a robot is trained to push a puck to a goal position. To enhance the agent generalization capacity, the authors introduce the concept of dynamic randomization that modifies a physical property of the world at each new episode, such as friction and damping of the puck or the mass of one of the links of the robot's body, to develop an agent able to adapt to the unknown real world settings. In (91), a robotic Shadow hand is tasked with solving a Rubik's Cube. In this case, the domain randomization technique is enhanced to deal with the additional image-based observations. Specifically, the agent trains in variants of the initial environment, where each variant has, in supplement to the alternative dynamic settings, different render properties. Using a wide distribution of texture, light and ambient occlusion parameters is a crucial component to ensure that the agent will be able to transfer its behaviour from simulation to the real-world.

3.6.3 Transfer between tasks

3.6.3.1 Reward-Based Transfer

The significant time required to train a DRL agent on a complex task has encouraged researchers to design strategies likely to increase the agent usability and avoid the exaggerated expenditure of computational resources on a single simulated task. While a part of these efforts was directed towards transfer to real-world configurations, several works set out to create agents which learned representations can be reused in another context. The works described in (3) fit in this context, as the authors demonstrate how policies trained in a self-play setting can display subtle behavior and that it becomes possible to observe emergent strategic behavior. Once trained, the agents are repurposed for another task. For instance, a humanoid agent that learned motor skills for a sumo task is then placed in a new environment where it must resist random wind forces and keep its balance. From a computational point of view, the neural networks used by the agent are not modified by the transfer and only the reward function will progressively affect the observed behaviour. Similarly, (2) conducts analogous experiments but in a multi-agent setting, namely a teamed *hide-and-seek*. This environment is genuinely complex as it features immovable walls as well as interactable objects that can be used by the hiding agents to create seekers-safe areas. After being trained on the original task, some agents are repurposed for alternative objectives such as simple construction tasks (which consist in putting blocks in predefined places). In the experiments described in (3), the transfer proves effective as the pretrained agent requires only a short amount of time to learn effective behaviour in its new environment. However, the additional tasks in (2) display a more nuanced result: the pretrained agent's performance and learning speed barely exceeds a baseline agent trained from scratch, thus demonstrating that transfer must be carefully planned beforehand to be beneficial.

3.6.3.2 Curriculum Based Transfer

The strategy presented above recalls the one used in curriculum learning environments. In curriculum learning (28), the idea is to progressively increase the difficulty of an environment. Starting in a simplified version of the task allows to easily design a reward that efficiently guide the agent towards the desired conduct. This approach has been employed in numerous recent works (15; 91; 111; 142). In (57), the authors establish that this strategy, which can be conceived as the transfer of previously acquired skills from one agent to a future version of itself, has numerous advantages. It simultaneously improves the generalization of the agent, prevents overfitting to a single and repetitive scenario (which can result in poor agent performance if the environment is slightly modified afterwards) and, in particular, allows the agent to reach a better performance level with less data, thus enhancing its efficiency.

3.6.3.3 Intrinsically motivated Transfer

Besides modifying the reward function or using a progressive curriculum to ensure agents adapt to new but related tasks, there exist alternative approaches to broaden the agent's strategies and indirectly set up the transfer of skills between tasks, thanks to this enlarged paradigm. One way to reach this goal is to prevent the agent being trained to focus on a single strategy to try and solve the task. In this view, multiple works building on the identified connection between RL and the information theory (87; 119; 153), rely on an additional objective function that involves the maximization of the action distribution entropy to ensure the agent learns various skills (43; 44; 122). When the agent's policies is able to display multiple competences, it is generally able to generalize better to new environments. Beside entropy-based incentives, other methods have been proposed to maximize environment exploration by the agent. (51; 83) use mutual information between states and actions to intrinsically motivate the agent, a concept that recalls the curiosity-based learning (4; 74; 117; 118). For instance, in (94), the curiosity is formulated as an error related to the agent's ability to predict the consequences of its actions in a high-dimensional observation space. The results of this formalisation suggest that curiosity provides a more efficient progression in an environment with sparse rewards, a better exploration that forces the agent to acquire new competences which in turn can be translated into generalization to unseen scenarios where the agent can reuse previously obtained knowledge and experience to progress faster. A similar technique is implemented in (10) that furthermore tackles the *noisy-tv* problem, a common issue when trying to predict observations in high-dimensional spaces, by expressing the curiosity error as the distance between the prediction of the next observation by the agent and the prediction of an other randomly initialized and fixed neural network (92).

The works presented in (31) take a substitute path, remarkable due to the absence of reward provided by the environment, a configuration commonly explored in the literature (72; 90). In this framework, an entropy-maximization objective function incites the agent to generate various trajectories for an additional goal description vector. Trajectories related to different goals must be as diverse as possible to minimize the entropy loss. In this case as well, the authors are able to demonstrate that this approach results in the unsupervised emergence of diverse skills. In addition, the skills gained during the thorough exploration of the environment give birth to transferable competences to slightly different tasks.

3.6.3.4 Meta-Learning

Meta-Learning for RL recalls the language models in NLP as it is in essence designed to be the basic understanding to a span of tasks. Specifically, this field develops techniques that directly optimize the adaptability of an agent. Although initially used for few shots classifications in supervised learning settings (116), there exist several notable attempts to apply these concepts to RL (25; 58; 141). Techniques for Meta-RL can be broadly classified in those classes:

- Memory-Augmented models: One of the first approaches developed consists in augmenting neural networks with memory in the form of LSTM (Long Short Term Memory), a type Recurrent Neural Network (25)
- Meta Learning algorithms: The works relying on this strategy set out to find ways to optimize simultaneously the model on a variety of tasks
- Distributions of environments: The agent is exposed to a multitude of environments to enhance its capacity to generalize.

An innovative example of Meta-Learning Algorithm is the MAML (Model-Agnostic Meta-Learning) algorithm, introduced in (32). The concept of MALM is that the optimization process *via* gradient descent should move the model in the parameters space to a saddle point so that task-specific fine-tuning is fast and efficient. A similar process is presented in (89), where the model is optimized by repeatedly sampling a task, training for multiple steps on it and updating the model parameters. This method is further improved in the CAML (Context Agnostic Meta Learning) algorithm (154) that decomposes the model into context parameters, intrinsic to a given task, and task-agnostic ones that are shared parameters. Doing so provides a wider distribution of tasks as the input and output dimensions are managed by the context parameters. Nevertheless, in practice, the task distributions considered in these settings are usually very narrow (149). For instance, a common evaluation of Meta-RL algorithms involves changing the maximal joint velocities of a serial robot on a manipulation task, while another one consists in choosing different running directions for simulated legged robots (32). These results are appealing from a theoretical point of view, but currently lack applicability as they are unlikely to be sufficient to enable robots to behave realistically in human environments. Nevertheless, the paradigm that motivates the creation of a central controller that can be quickly adapted to different environments is highly interesting and is the main pillar of transfer in hierarchical settings

3.6.3.5 Transfer in Hierarchical Settings

Generally, transfer in hierarchical settings implies jointly learning a set of skills and a meta-controller, introduced as *Options* formulation in (131). This formulation assumes that the *Options* are given, but this concept has been actively explored by the research community since then. Consequently, it is now usual to learn these *options* as well as the policy that selects them (1). Some works propose a decomposition of the considered task into subgoals (33; 35; 49; 140). Other techniques break the task between low-level and high-level control, as in (98) where the authors partition the locomotion components and use the higher-level controller to provide goals to the lower-level controller. In this method, both controllers are learned simultaneously and only rely on model-free modelling, that is without knowledge of the equations of motion, nor character kinematics. An interesting aspect of this method is that the hierarchical structure allows interchangeability between components, thus increasing the usability of the agent.

3.7 Knowledge representation in Transfer Learning

Attempts to transfer knowledge are also represented in the unsupervised learning paradigm. In fact, the relationship between unsupervised learning and Transfer Learning might even appear more natural as one of the main objectives in this paradigm is to be able to train models whose optimization leads to a clear understanding of how data were generated. Indeed, when the model captures efficiently the data structure, it is in particular able to provide relevant representations for subsequent usages, which proves highly useful in neural-network based controllers (68; 69). As such, the works presented in (42) introduce the World Model approach where the authors leverage a three-part sequentially trained model that was able to overcome the previous state-of-the-art on the challenging *CarRacing-V0* Gym environment (9). Most of this performance relies on the first module of this model: the encoding part of an AutoEncoder architecture trained on a reconstruction objective, using the raw pixels from the environment. The small dimensionality of the encoder output forces the whole network to learn meaningful state representations (that is, focus on the input most salient features) that proves usable for downstream tasks. As the model is trained in a sequential manner, it is possible to imagine that this module could be paired to another controller to solve a different task, thus illustrating the transfer. There exist numerous examples of AutoEncoder architectures being successfully applied to create meaningful representation of data. Using Variational AutoEncoder (11; 62) (VAE) is also frequent as the additional objective function in these architecture enables a better control over the encoder latent representations (14; 27; 46; 76; 144). These techniques can enable transfer beyond the usual agent-agent paradigm. In particular, the method developed in (75) takes advantage of this property to project complex multi-DoF expert, potentially humans, trajectories for a given task in a small latent space. The VAE properties ensuring that the representation is liable and consistent, the authors of this work demonstrate that it can be used in an assistive robotics setup to ease the control of multiple DoF robots for subsequent users, requiring only one or two DoF. The GAN framework also presents specific appeals for the transfer of skills. Since the introduction of Neural Style Transfer (34), a model that transfers the painting style of a given artist to another picture, efforts were invested to establish the capabilities of transfer in the GAN paradigm. One of the pioneer approach, the CycleGAN (152), demonstrates unpaired image to image translation, that is, translate an image from domain A to domain B, was a feasible task that yielded highly qualitative results, see Figure 3.9. Using a similar principle, the AVID (Automated Visual Instruction-following with Demonstration) method (129) replaces the human operator in the videos of a task execution by a given robot. Then, the authors demonstrate that it is possible to quickly train a model-based RL policy that manages the robot transitions between translated states.

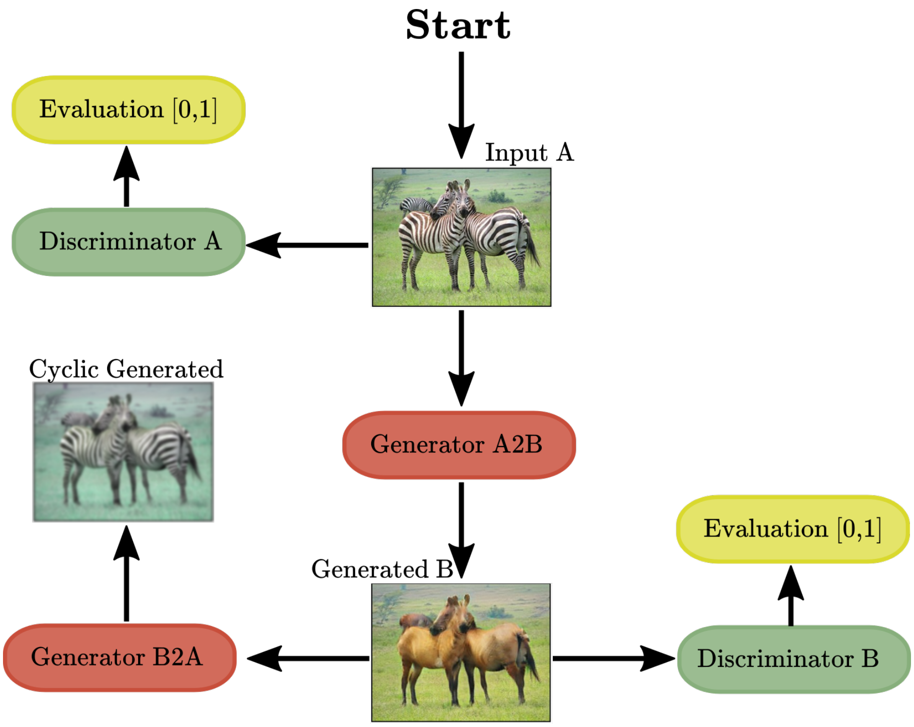


FIG. 3.9. CycleGAN training method.

3.8 Conclusion on Transfer Learning

The above section introduced the most common and popular approaches of Transfer Learning and fine-tuning in recent deep learning works. Supervised settings offer a rather straightforward framework to apply these techniques and transfer the embedded knowledge representation from a model trained on a given task to another objective. Due to the model architecture and dimensionality bottleneck. Realizing a similar process for RL in control tasks is less direct. Indeed, as can be seen from the recent trends, the community, fully aware of the computational inefficiency of current algorithms is actively researching ways to fully take advantage of the representations learned by a trained agent. Notable approaches to reach this goal include exposing the agent to a wide variety of environments, maximizing its ability to explore, using meta-learning algorithms or a hierarchical organisation of skills. All of these paradigms present undeniable advantages and strong appeal. Nevertheless, each one also focus on a single agent with a fixed morphology. However, there exist numerous real-world cases that would rather benefit from transferring knowledge from one entity to a distinct one. Consequently, the methods proposed in this work are designed to tackle this seldom considered transfer situation.

Chapter 4

Task-Centered Transfer

Contents

4.1 Universal Notice Network

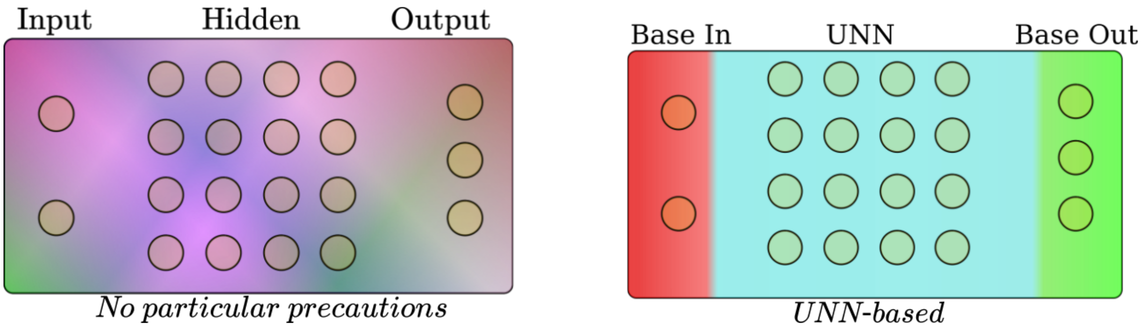
This section focuses on the TAsk-Centered approach for transfer learning, one contribution of this thesis. It goes over the proposed Universal Notice Network framework in depth and explains how this technique is relevant with current state-of-the-art. Later, it is provided various application scenarios allowing us to demonstrate the UNN adaptability.

4.1.1 Principle

The Universal Notice Network (UNN) is a TAsk-Centered transfer learning technique. The main motivation of this family of methods is to tackle the fact that backpropagation optimization, unless specifically constrained, tends to result in entangled knowledge representations, as schematized in Figure 4.1a. Indeed, as explained in (11), unless a specific loss function is dedicated to disentangling knowledge, neural networks final weights composition will be heavily influenced by initial weight distribution and examples used for backpropagation. This issue can be partially mitigated when the networks have a big enough capacity and present dimensional bottlenecks. Specifically, as detailed in Section 3.3, this can be easily observed in usual CNN architectures, such as (128; 135; 146) used for classification tasks. In these configurations, works have shown that the main feature representations are essentially located within the convolutional layers, the last linear layer being only used as a translator to the resulting prediction. As a result, it is conceivable to remove the last fully connected layers without losing general information. Similarly, in recent NLP models, specific modules are dedicated to word embeddings, for instance, and can be detached and passed to other models in a straightforward manner as a feature extractor model. In most RL configurations, the architectures considered are shallow and do not present any dimensional bottleneck, hence leading, if no particular precautions are taken, to entangled knowledge representations. Furthermore, due to the environment stochasticity, the action sampling and randomized weight initialization, it is difficult to predict the parameters trajectory in their space during the training session. While this entangled representation does not prevent the agent from learning, it however makes it difficult to determine which part of the network is responsible for a given behaviour. As shown in Section 4.2.1, this rules out the *vanilla* transfer learning approach that consists in replacing the latest layer to fit the new problem dimension.

Consequently, the UNN is a framework that implements the idea of knowledge segmentation between the agent and the task. The main UNN principle is derived from the idea that any agent can achieve a task by following the right set of instructions, assuming that the agent is able to move adequately.

To better understand the UNN philosophy, let us consider the following analogy: a consumer acquires a furniture in a store and upon receiving it, expresses the desire to assemble the object. As expected, the product is delivered with a notice that expresses the steps needed to successfully setup the product. To be usable by the customer, the notice needs to be generic enough to avoid direct references to elements that the client does not possess



(a) Straightforward training setting.

(b) UNN setting.

FIG. 4.1. Knowledge location comparison.

or requires movements that he could not be able to do, for instance requiring too much strength or flexibility, as shown in Figure 1.1

In this view, if an efficient task model is generated by the agent, it then becomes possible to transfer this task model to another differently structured agent and allow it to perform the considered task. The challenge in this process lies in the preservation of the task model parameters from the agent morphology influence, ideally reaching a segmentation as displayed in Figure 4.1b. The more perceptible the influence is, either directly from gradient updates or from the agent behaviour, the more likely it is that the task model be corrupted, hindering its transfer efficiency, thus requiring a fine-tuning step. On the other hand, a well-preserved task model increases the chances for immediate plug-and-play behaviour, sometimes allowing zero-shot transfer (transfer without retraining phase).

4.1.2 The UNN workflow

In this paradigm and in order to either construct the UNN or simply plug it in an already existing pipeline, it is crucial that the agent already possesses a way to comply with the UNN instructions. Specifically, this implies that the agent must be able to move in a coherent and relevant way. To satisfy this last constraint, the agent is pre-trained on a primitive task. The primitive task constitutes a highly important step and must be designed to allow the more thorough exploration of the observation and action spaces the agent is likely to encounter when paired with the UNN. The objective of this phase is to enable the agent to acquire the basic motor skills that are relevant to the tasks it may be expected to fulfill. As explained later in Section 4.2.6, this step heavily influences the UNN generation and can ease the learning process. The UNN is then trained using this basic motor skill module. In this phase, the optimization process focuses on the UNN module, leaving the motor skill part unaffected by gradients. This first stage is displayed in Figure 4.2.

Once trained, the UNN module can be transferred to an agent structurally different from the one used to create it and consequently enable this new agent to perform the task with minimal to no prior interaction with it. In Figure 4.3, stage I corresponds to the initial choice of task and agent. The computation pipeline is created in stage II by pairing the

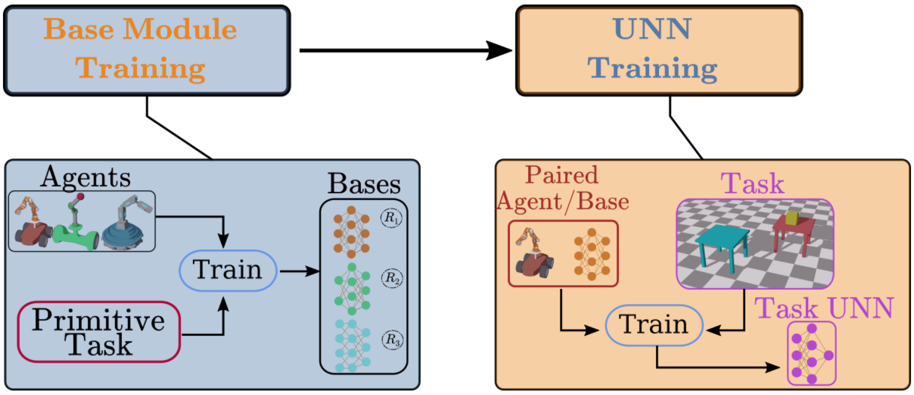


FIG. 4.2. The original UNN Pipeline staged training first creates a primitive agent controller (in blue box) for a chosen agent configuration. The UNN for the task relies on the generated primitive motor skills to learn a successful policy (in light red box).

trained task UNN and the base module of the performing agent (this agent is different from the one that initially constructed the UNN). This makes possible the achievement for this task by the new agent as shown in stage III. The next paragraph describes the computational pipeline set up, while the rest of this section provides details on the modules creation.

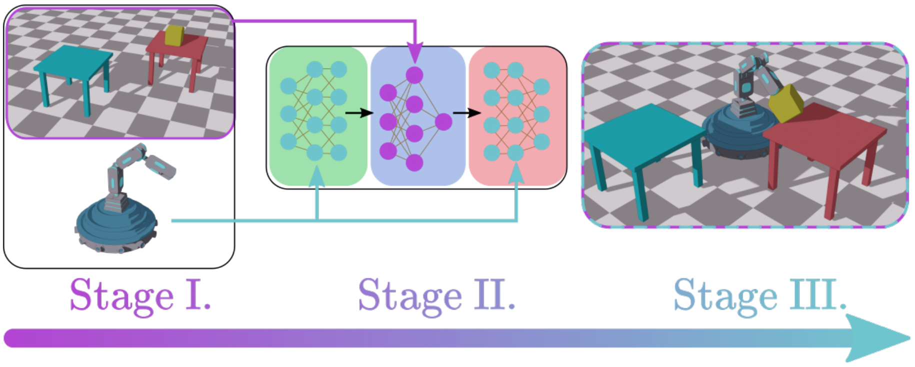


FIG. 4.3. The successive steps for deploying the transferred UNN to a new configuration.

4.1.3 The UNN Pipeline

In practice, the UNN pipeline is a model composed of three sub-modules, $m_i^r, m_u^{\mathscr{T}}, m_o^r$, as shown in Figure 4.4. The first and the last parts of this model m_i^r, m_o^r, respectively the input (green) and the output (red) modules, are specific to the robot r, while the center model $m_u^{\mathscr{T}}$, the UNN (blue), is designed to be robot-agnostic and specific to the task $\mathscr{T}$. Specifically, the state vector that is observed at each timestep, can be split into two parts $s^{i,r}, s^{\mathscr{T}}$, respectively holding data intrinsic to the considered robot and task-related information, independent from the agent. The agent specific state $s^{i,r}$ is used by its input module to compute a state agent representation:

$$s^{i',r} = m_i^r(s^{i,r}) \tag{4.1}$$

The UNN (i.e. the task module), conditioned by the task observation vector $s^{\mathscr{T}}$, uses the processed agent representation vector to compute a vector that can be seen as a high-level instruction:

$$o^{\text{out}} = m_u^{\mathscr{T}}(s^{\mathscr{T}}, s^{i',r}) \tag{4.2}$$

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

$$a^r = m_o^r(o^{out}, s^{i,r}) \tag{4.3}$$

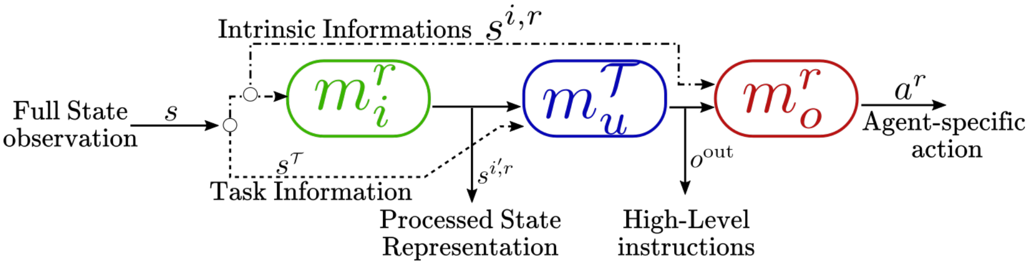


FIG. 4.4. The UNN Pipeline is composed of three stages: the input and output bases (in green and red) are specific to the robot, while the UNN (blue) is task-related.

In the cases discussed below, the UNN and the output module are neural networks, but in practice, there is no theoretical restriction on the computational model used, as long as the vector returned by the modules has the expected dimensionality. From a functional point of view however, to ensure that the UNN is compatible with transfer, it is necessary to design the vector $s^{i,r}$ that is passed between the UNN and the output module in a way that ensures the Markov property (Section 2.5.1) is satisfied: it must encompass enough information for selecting the next action without needing to consider additional previous steps. The same logic is applied to the vector passed between the input module and the UNN.

4.1.4 Base Abstracted Modelling

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

$$\begin{aligned} &\text{find} && m_i^r, m_u^{\mathscr{T}}, m_o^r \\ &\text{such as} && g_{\mathcal{T}}(\mathscr{M}) \leq 0 \\ &&& g_r(\mathscr{M}) \leq 0 \\ &\text{With:} \quad && \mathscr{M} = f(a_r) = f(m_o^r(m_u^{\mathscr{T}}(s^{\mathscr{T}}, m_i^r(s^{i,r})), s^{i,r})) \end{aligned} \tag{4.4}$$

However, due to the notable RL tendency to fall into local minima, the UNN is likely to discover a successful strategy for solving the task that depends on its body configuration (for instance, blocking an object between two articulations). In these deceptive cases, the constraint related to physical limits in Equation 4.4 may no longer be respected. While this is not an issue for the current agent, it is detrimental for the efficiency of a future transfer to an agent with a different structure. To ensure that the UNN constraints are not entangled with the agent ones, the Base Abstracted Modelling (BAM) is introduced (85). This approach relies on an environment modification that assimilates the robot to its effectors, thus creating a virtual robot, as shown in Figure 4.5. On the formulation side, we set the base models m_i^r, m_o^r as identity functions and provide directly the UNN output to the command. These two modifications release most of the constraints that would have been distilled by the pre-trained base and prevent the UNN from learning a configuration-specific strategy, resulting in the creation of an UNN closer to a model-agnostic setting.

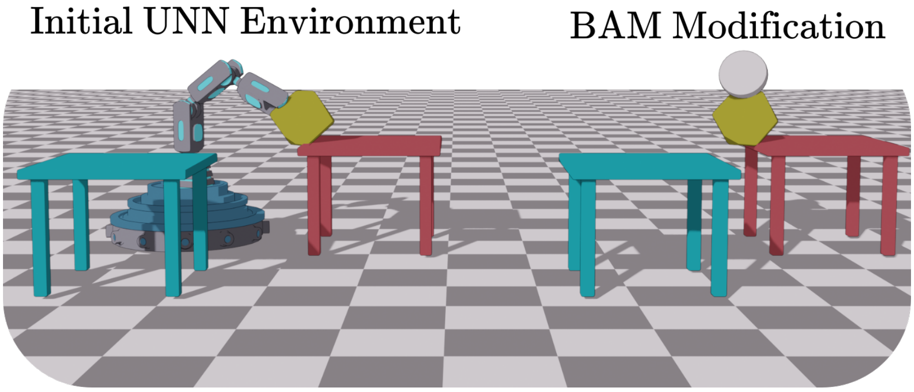


FIG. 4.5. The BAM version of the environment prevents specific behaviour from leaking in the task model, leading the UNN closer to a bias-free logic.

4.1.5 Building and using UNN modules

The motivation behind the UNN approach is to ease and enhance the efficiency of skills transfer between agents of different configurations. We provide here a possible workflow description relying on the UNN, and shown in Figure 4.6.

Let us imagine that a set of robots with different configurations and a set of tasks achievable by these robots are available. Now, consider the case where a user wants a robot r to perform a given task $\mathscr{T}$. Given the training segmentation between the control model and the task requirements, the UNN model is agent-agnostic, meaning any compatible robot could have been used to create it. Hence, once the UNN module exists, it can be shared with agents presenting other morphologies. It opens the possibility for a user to pair any robot up with its module with a UNN relevant to the current task and carry on with the realization (green path in Figure 4.6). If no robot modules are available, it is possible to generate them, using a primitive task: this is the robot module creation (black path in Figure 4.6). Alternatively, a robot module can be created using a given task and its paired pretrained UNN (red path).

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

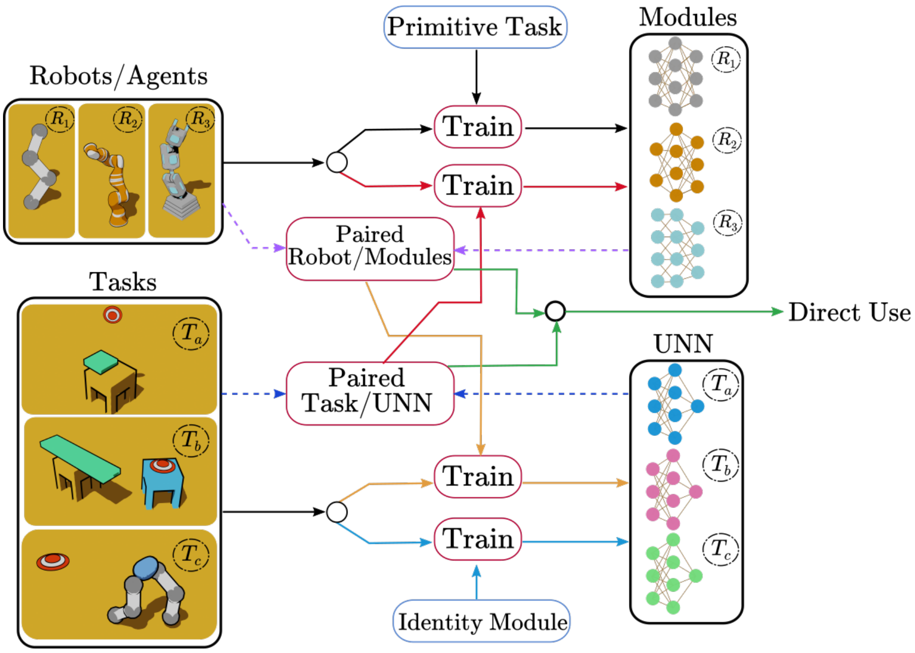


FIG. 4.6. The UNN workflow: Various solutions for creating and using modules and UNNs. Dashed violet paths are to pair the robot with its module. Dashed blue paths are to pair the task with its UNN. Black and red paths are to create new robot modules. Orange and blue paths are to create new task module (UNN).

two ways: relying on its trained robot modules to train the UNN (orange path) or using Identity modules such as the BAM framework (blue path), explained in Section 4.1.4.

4.2 Experiments

In this part, we focus on the experiments realized to validate this approach. Firstly, we demonstrate that *vanilla* transfer, in the sense of CNN models, is not a reliable technique when considering RL agents, even in very close conditions. Motivated by these results, a serie of experiments relevant to the UNN are presented. The initial primitive task, used to enable the agents to acquire desired motor skills is introduced as well as the agents considered for these manipulations. Afterwards, the UNN pipeline is applied to a wide spectrum of situations, ranging from simple manipulation to more complex co-manipulation and collaborative tasks. For each configuration, the UNN performances, both in learning and transfer, are compared with a state of the art algorithm and its *vanilla* transfer as a baseline. Finally, an alternative usage for the UNN is introduced. Specifically, this last approach evaluates the possibility of recovering a motor skill through the trained UNN module. Additional results covering mobile manipulator configurations are available in the Annexe A.

4.2.1 Vanilla transfer

In order to motivate further the TAsk-Centered approach, we design a toy experiment to underline the liability of *vanilla* transfer in RL framework. In this view, we consider two exactly similar agents that are trained on a unique task. Given that this experiment aims at demonstrating that unconstrained backpropagation does not structure knowledge in common RL architectures, the next step is to evaluate how *vanilla* transfer would affect the two trained agents. Specifically, two additional configurations are created by replacing the hidden layers of an agent in the other one, as shown in Figure 4.7. The performances of these two are then measured on the same task.

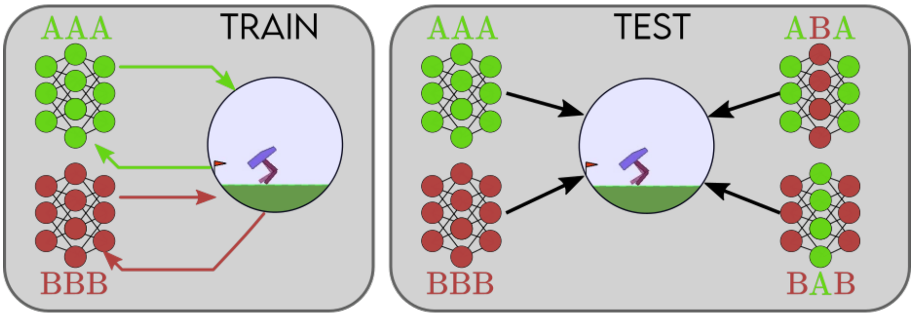


FIG. 4.7. Exchanging model layers between similar agents.

On the realisation side, the task selected for this experiment is **BipedalWalker-v3**, a common benchmark for RL. In this environment, the agent controls a biped and can apply torques to its joints. The environment provides observations to the agent in the form of a 24-dimensionnal vector that describes the agent joints states and the reading from several raycasting. Upon observation, the agent returns an action vector of 4 components, that act as joints speeds. The goal is to prevent the biped from falling, which results in a -100 reward, while bringing the agent towards the end of the path, granting a +300 reward. In order to ensure the agent has a responsible control, small negative reward proportional to the torque magnitude are given at each time step. **BipedalWalker-v3** belongs to the Gym suite control tasks and even though this environment is among the most complex of this suite, decent policies can easily be created by a recent RL algorithm such as PPO or TD3. In this case, the policies considered are neural networks featuring one hidden layer of 64 units. Figure 4.8 displays the mean reward evolution along training. As can be seen in this Figure 4.8, the mean reward quickly hovers around 200 points, while a baseline random agent usually gets cumulative rewards that do not exceed -90 (9).

It is then possible to apply the *vanilla* transfer between these two agents, for all eight possible configurations. Figure 4.9 displays the evaluation performances of all settings.

As can be observed, both genuine configurations (in blue and gray) exhibit a wide distributions of results, and even though these policies have inevitable failed runs, they are also able to reach scores above the 200 points threshold. This contrasts strongly with all recomposed policies, for which none of the runs exceeds the -50 points threshold. In this case, although the **Bipedal Walker** presents a considerable challenge, it would have been

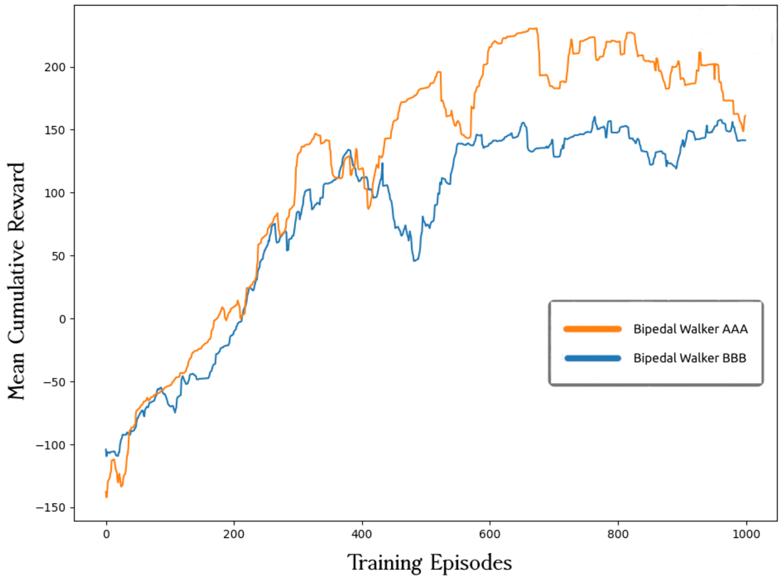


FIG. 4.8. Mean reward evolution on BipedalWalker-v3 environment.

reasonable to expect that at least a fraction of the performance would have been transferred, given the fact that both agents reach high scores and share the same morphology. Consequently, it appears that *vanilla* transfer is not efficient in the RL framework and that a more efficient approach would be a valuable contribution.

4.2.2 Learning setup

As explained, the UNN method segments the task knowledge from the agent control parameters. While the control part can be defined through various methods (for instance, through analytical formulations), depending on the user needs, we focus here on the least robust cases where this part is learned through a Reinforcement Learning process. We present various experiments, detailing the UNN capabilities in manipulation tasks, co-manipulation (both in regular and BAM framework) and a cooperative walking task. Specifically, we show that it is possible to repurpose a robotic arm for various tasks and that the knowledge gained using this configuration can be used with very little to no retraining by other agents configurations. Our agents are trained using Proximal Policy Optimization (PPO) (122). As for the network architectures, both the UNN policy and the output robot module network have two hidden layers with respectively 128 and 64 units, with TanH activation function. Our reference baseline, the *vanilla* PPO policy has three hidden layers of 128 units, again with TanH non-linearity. Given that all our robots have a serial structure, we propose to affect a Forward Kinematic analytical model to the input robot module to compute $s^{i',r}$, the processed input state, such that $s^{i',r} = m_i^r(s^{i,r}) = \text{FK}(s^{i,r})$. To evaluate transfer efficiency, both regarding learning and final performances, we consider four configurations:

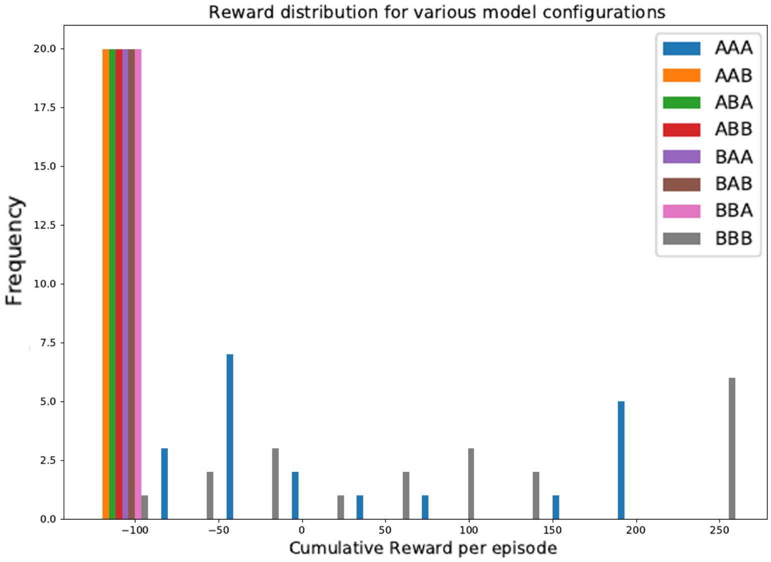


FIG. 4.9. Comparison of all configurations. Genuine configurations are able to reach high scores while all recomposed one fail to exceed a very low threshold.

- **PPO**: A baseline *vanilla* PPO agent is trained from scratch

- **PPO Transfer with fine-tuning**: A new agent is initialized with the previously learned weights from the *vanilla* agent. In this case, the hidden weights are fixed during training and the optimization process affects only the input and output layer weights.

- **UNN**: The UNN is trained from scratch while being paired with a trained robot module or in its BAM version

- **UNN Transfer**: The previously learned UNN is transmitted to another robot, using its own trained robot modules. When the training percentage in the various table results exceeds 0%, it means that the bases are fine-tuned for this percentage of the initial training time. Specifically, the output base weights are fixed during retraining and only the UNN module will be updated by training.

4.2.3 Controlled Robots

The experiments presented in this chapter aim at demonstrating the UNN method adaptability by setting up transfer of tasks between several robots and thus show that this approach is able to create a module encompassing knowledge independently from the agent accomplishing the task, in order to later have the possibility to transfer this knowledge module to an agent with a different configuration.

In this view, Figure 4.10 shows the robots used to embody this transfer, along with their Degrees of Freedom (DoF) (drawn as colored arrows). Some architectures are inspired

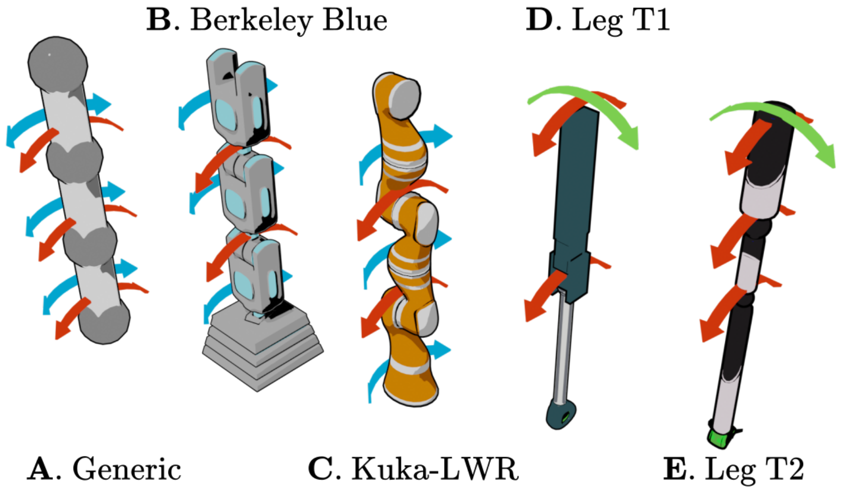


FIG. 4.10. From left to right: Generic-3 robot, Berkeley Blue, Kuka-LWR, Leg Type 1, Leg Type 2.

from real-world robots, such as the Kuka-LWR (5 DoF) or Berkeley Blue (5 DoF). We also introduce the Generic-3 robot, a 3 joints with 2 DoF for each joint configuration, the Generic-2 robot, which exhibits a similar structure with only two joints and an architecture declined into two variants (Leg Type 1, 3 DoF, and Leg Type 2, 4 DoF) that was used in the Biped environment.

4.2.4 Base Manipulation: Reacher

The reaching task is presented in Figure 4.11 and consists in setting the position of the end-effector to a desired position. This is a fundamental and particular piece in our framework, since most of the experiments will require the agent to have mastered this skill. That task is considered as the primitive task introduced in Figure 4.6. However, while seemingly very basic, an agent trained with a reward depending solely on the distance between its effector and the target very rarely yields a policy with plausible/desirable behaviours. Most of the time, that kind of reward function ends up into a policy with unexpected behaviours at best and unstable at worse (Figure 4.11, right-most robot). Hence, to avoid this drawback of reinforcement learning, we take advantages of various insights from (100) and we propose the following MDP for learning:

- $s^{\mathscr{T}} \in \mathbb{R}^7$: Vector from the end-effector to the target and target orientation (as quaternion), expressed in world frame

- Reward function R_t: Inversely proportional to the distance between current agent configuration q_a (Figure 4.11, left-most robot) and a configuration given by a state-of-the-art method q_r, here (130) (Figure 4.11, central robot). Thus:

$$R_t = -(1-c) \times \alpha \times ||q_a - q_r|| + c \times \beta \tag{4.5}$$

where c is a Boolean value that is set to 1 when the distance between q_a and q_r is below a user set threshold. In these cases, the agent receives a small postive reward β. Otherwise, the reward is inversely proportional to the distance between the agent configuration and the target configuration, with α a scalar used for reward normalization. The q_r configuration is the main leverage used to induce bias within output module. Indeed, in this specific task, the UNN is transparent, acting as the identity function. Thus, given that training focuses on the output module, then by defining wisely the q_r configurations in this task, it is possible to generate output modules that, for instance, can conditionally behave in a way that recalls inverse kinematic models constrained through pole targets. This feature is particularly useful and its usages are more thoroughly detailed in Section 4.2.6.

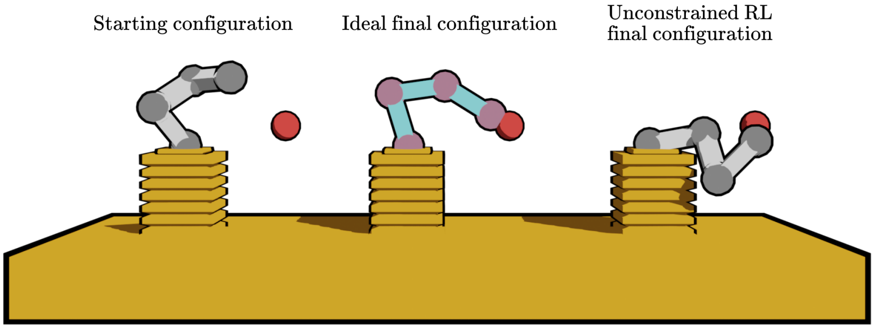

Starting configuration
Ideal final configuration
Unconstrained RL final configuration

FIG. 4.11. Reacher Task.

Reaching results are presented in Figure 4.12, that displays the filled area between the worst performance and the best along training for the six agent configurations over 3 different initial seeds. In this figure, it is possible to observe that even though the initial agents performance displays a wide variance, their overall evaluation with respect to the environment is improving along training and reaches levels above zero, implying that the agents are able to quickly position their effector close to the target and follow it during the episode while satisfying the configuration constraints.

4.2.5 A lonely manipulation: Tennis

This task consists in a self-play in a tennis-like setting, see Figure 4.13. A paddle effector is considered as the end-effector of a robotic arm and the goal is to maximize the number of times the ball bounces against the wall and the episode ends up whenever the ball drops below a preset height threshold. Thus, the MDP has the following components:

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

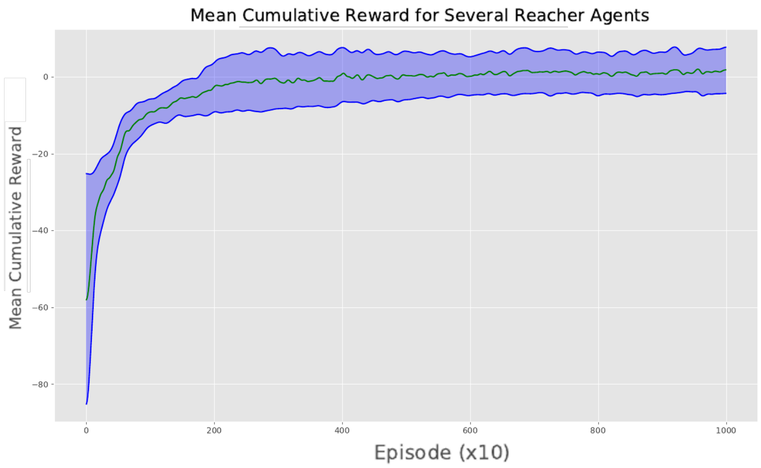


FIG. 4.12. Mean reward along training on the primitive reacher task.

- Reward function:

$$R_t = \begin{cases} 1 & \text{if contact} \\ 0 & \text{else} \end{cases} \tag{4.6}$$

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

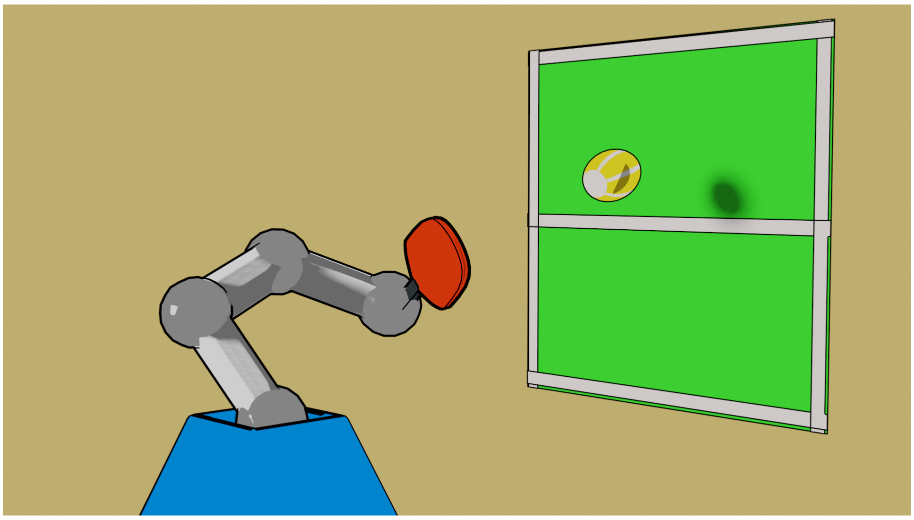


FIG. 4.13. Tennis Task.

Figure 4.14 displays the performance curves during the training and transfer phases. In this case, the original PPO policy and the UNN are trained for 1 million timesteps controlling the Kuka robot. It clearly appears that the UNN policy (dark blue) is superior by an important margin to the *vanilla* PPO policy, both in terms of learning speed and final performance. We detail various mechanisms fueling this phenomenon further. Once training is over for those two policies, their trained models are transferred to a new agent

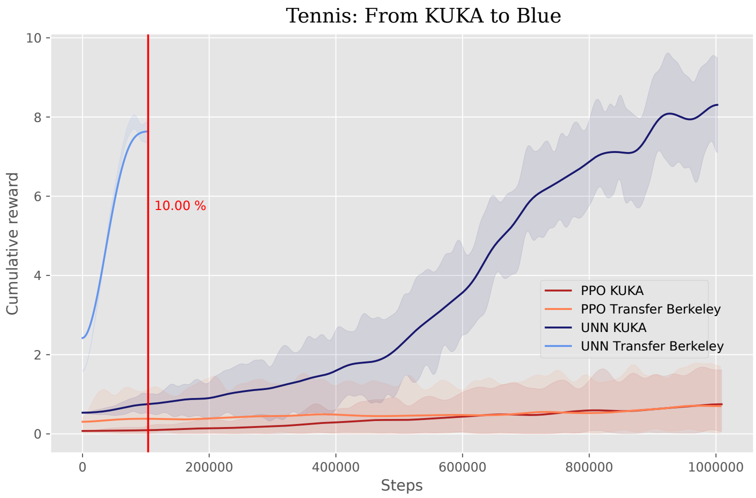


FIG. 4.14. Learning performances in the Tennis environment. PPO and UNN are performed using Kuka robot, transfer to Berkeley Blue robot.

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

		Config	**Training**	**Performances** (in bounces)
UNN	Baseline	Blue	100 %	19.01
	Original	Kuka	100%	18.74
	Transfer to	Berkeley Blue	0%	11.88
	Transfer to	Berkeley Blue	10%	17.56
	Transfer to	Generic 3	15%	18.12
PPO	Baseline	Blue	100%	7.54
	Original	Kuka	100%	7.89
	Transfer to	Berkeley Blue	0%	0.86
	Transfer to	Berkeley Blue	100%	6.81
	Transfer to	Generic 3	100%	7.57

TABLE 4.1. Tennis Summary.

Once training is over, we evaluate the final policy performance by averaging the number of bounces against the wall for 100 episodes. These results are detailed in Table 4.1.

This table and the subsequent tables in this chapter all share the same organisation. The first column gives the training setting, whether the policy considered is the original or a transferred one. The second column, entitled configuration, details which robotic arm or dual-arm was used for this simulation. The third column precises the percentage of total training steps used for the performances of this policy, listed in the last column. In this case, these results confirm the performances predicted by the learning curves, as well as the transfer efficiency. In particular, we see that after training for 1 million timesteps (100%), the UNN policy is able to bounce the ball over twice as much as the *vanilla* PPO policy. Also, we see that with no retraining, the UNN controlling the Berkeley Blue robot outperforms both the original PPO with the Kuka as well as the transferred PPO to the Berkeley Blue robot. In Table 4.1, a transfer toward the generic case has also been added, further emphasizing the domination of the UNN on this task.

The above results are prone to raise several questions about the reasons why UNN agents outperform *vanilla* PPO policies. The visual observation of each agent performance as well as the training context of its module reveal important qualitative facts. In this case, the UNN output module is a network trained on a reacher task, using a reward function that encourages the agent to propose a solution that respects certain constraints, as described in 4.2.4. Hence, even with randomly initialized UNN module weights, the agent initially returns joints configurations that are close to the ones needed to be able to easily bounce the ball back towards the wall. It is a good starting point and, through the restrictions it brings, helps the UNN policy find solutions in the current MDP. On the contrary, the *vanilla* PPO policy has no particular reason, when initialized, to propose those expected configurations and freely explores its action space. Thus, the policy may potentially receive its first positive rewards while being in a non-optimal position, see Figure 4.15, that will then see its probability increased due to the RL-optimization process. This is further exacerbated when the reward function is sparse and requires an important amount of tweaking to ensure the *vanilla* PPO policy receives its first rewards in a configuration that would allow it to keep improving.

Through the scope of the experiments led in this environment, it appears that the UNN methodology can offer non-negligible advantages in a transfer setting, as it was designed to do. Furthermore, it also provides leverage in the initial learning process as the embedding of soft constraints in its output module can ease the learning process of the UNN policy, which can ultimately lead to higher quality transfers. This second feature recalls the CV and NLP pretraining step and it is further explored in the next environment: the Biped Walking.

4.2.6 Biped walking

To broaden the range of applications using the UNN method, we introduce the walking biped task. In this environment, we want a biped robot, composed of a pelvis and two legs, to walk across a platform, see Figure 4.17. We assimilate this application to the centralized control of two reacher robots and, similarly to the co-manipulation task in

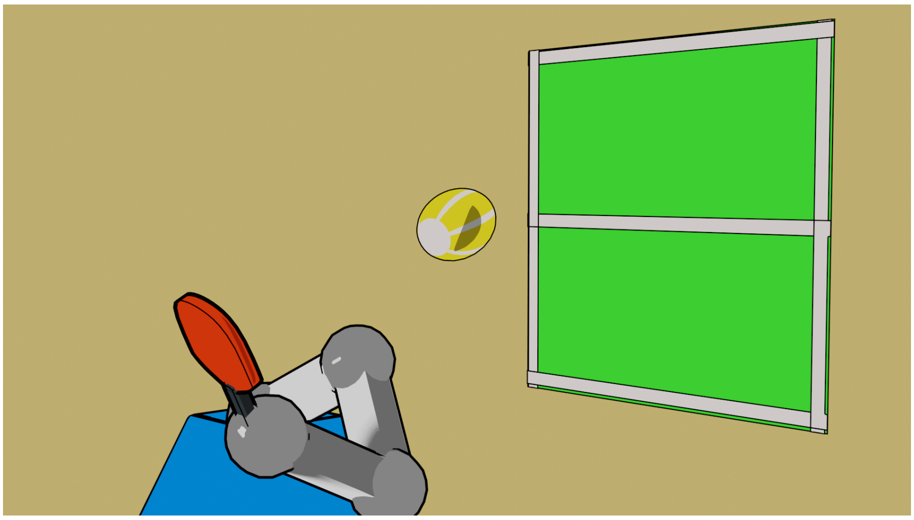


FIG. 4.15. Undesirable configurations that ultimately return positive rewards are likely to be considered as valuable states by the agent.

Section 4.2.8, the pipeline consists in a single UNN setting target positions for each leg, as shown in Figure 4.16.

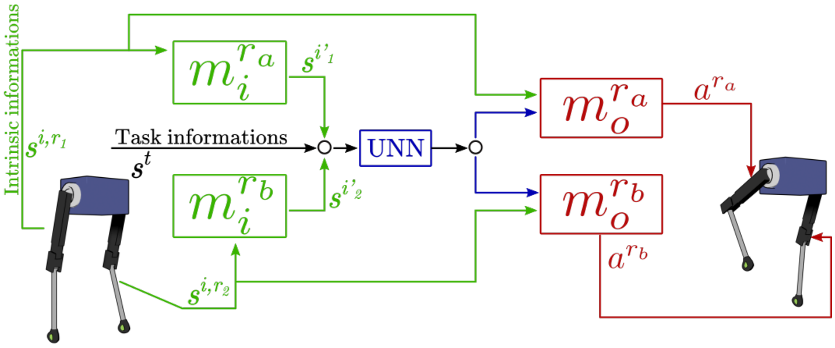


FIG. 4.16. Dual UNN Architecture.

The MDP for the walk UNN described below requires a slightly more complex reward function:

- $s^{\mathscr{T}} \in \mathbb{R}^{11}$: Pelvis linear position in a frame located at the center of masses (CoM), pelvis velocity and angular velocity in world frame, pelvis height from the ground, pelvis lateral offset from the middle of the platform.
- The reward R_t is composed of various terms, that is: $R_t = R_f + R_l + R_h$, with:
  - the pelvis speed along the forward axis: $R_f = \max(V_f, 0)$ where V_f is the pelvis speed along the forward axis,
  - the lateral offset between the pelvis and the forward axis, to encourage the agent to walk in a straight line: $R_l = \exp(-|P_l|)$ where P_l is the lateral pelvis position regarding the forward axis,

- the vertical offset between the pelvis and the target height (initial pelvis height, in practice), to prevent the agent from jumping:

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

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

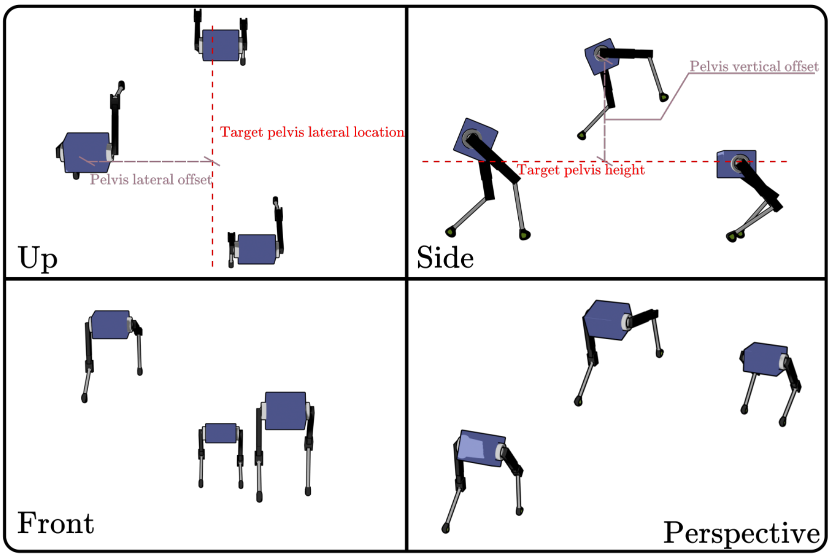


FIG. 4.17. Biped walk task.

Analyzing the curves from the Biped runs, in Figure 4.18, seems to yield partial contradictions to the precedent conclusions. Firstly, the *vanilla* PPO policy mean reward evolution along training clearly exceeds the UNN mean reward. And, while the transferred PPO policy does not clearly benefit from the transfer, apart from a steadier evolution, its mean reward grows also faster and reaches higher levels than the UNN mean reward. Even though the transfer in the UNN case yields an important initial advantage for the new policy, the final mean reward stays way below both *vanilla* PPO approaches. At test time, we compare the agent performances by measuring the average distance (expressed in meters) covered by the considered agent over 100 episodes, see Table 4.2. In this case, interestingly enough, the UNN agent and its transferred version outperform both PPO versions.

In this specific environment, the output module pretraining influence is even more pronounced. Indeed, in this case, the output module was trained again on a reaching task, but this time, we enforce through the reward function a bias for a knee back semi-folded leg configuration, as shown in Figure 4.19. Hence, when paired with the UNN, the module interprets the actions ordered by the UNN through this particular bias, which leads to a very different gait evolution between the UNN policy and the *vanilla* PPO policy which freely accesses to all its action space. As a result, the trained *vanilla* PPO policy exhibits a

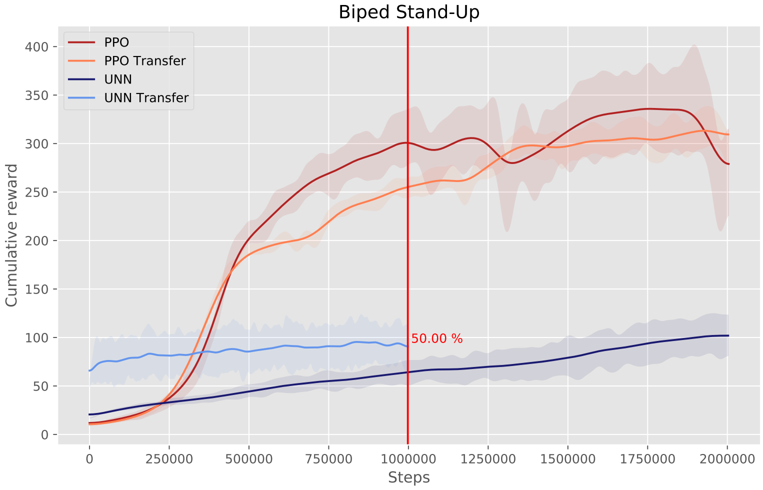


FIG. 4.18. Learning performances in Biped Walk environment. PPO and UNN are performed using 2 legs of Type1, transfer to robot composed of Leg Type1 and Leg Type 2.

		Config	**Training**	**Performances** (in meters)
	Baseline	Leg Type 1/2	100%	41.88
	Original	Leg Type 1/1	100%	49.74
	Transfer to	Leg Type 1/2	0%	21.02
UNN	Transfer to	Leg Type 1/2	30%	44.55
	Baseline	Leg Type 1/2	100%	32.91
	Original	Leg Type 1/1	100%	29.63
	Transfer to	Leg Type 1/2	0%	2.90
PPO	Transfer to	Leg Type 1/2	100%	31.67

TABLE 4.2. Biped Summary.

non-symmetrical gait, in which one leg is always forward, while the other leg stays behind, see Figure 4.20a. This moving position allows the *vanilla* PPO policy to gather at least two reward terms out of three, namely the offset reward R_l and the height reward R_h, thus explaining the higher PPO score in Figure 4.18. However, as shown in Table 4.2, this approach does not optimize for the walking distance, which would require a dedicated ponderation of the reward terms.

Indeed, this policy is able to move slowly forward by giving quick impulses to its joints, in a stutter-like manner. On the contrary, the trained UNN policy is much more symmetrical and alternates between small jumps and walk to move forward, see Figure 4.20b, resulting in a larger distance covered, see Table 4.2.

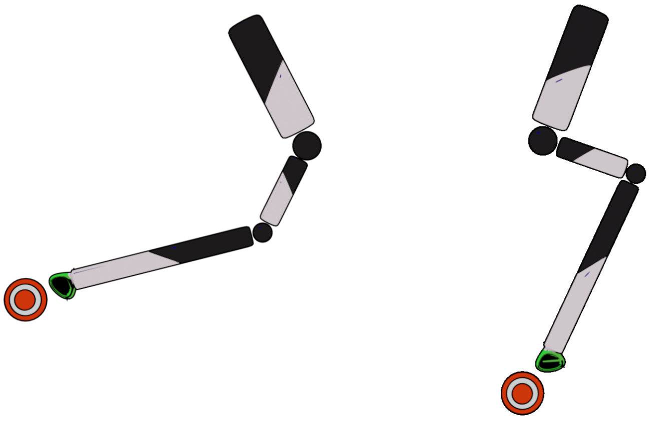
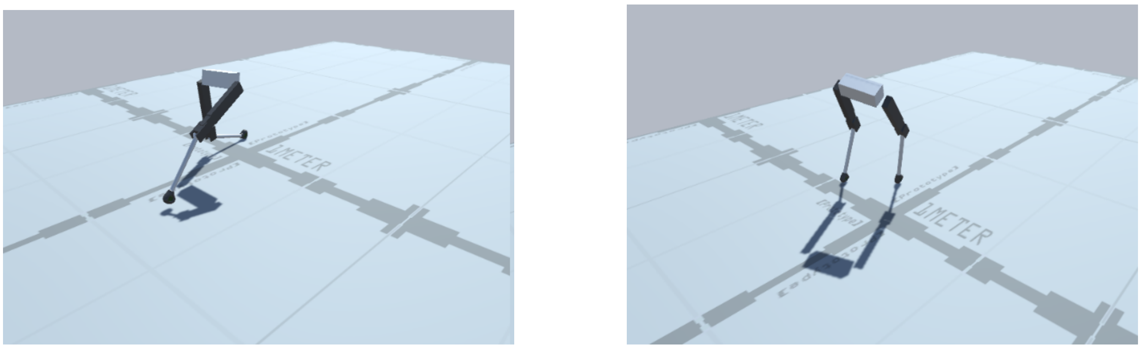


FIG. 4.19. Left: Free configurations. Right: Enforced semi-folded configurations.



(a) Straightforward training setting.

(b) UNN setting.

FIG. 4.20. UNN pretraining can nudge the policy towards a more human-like gait.

4.2.7 Cooperative Manipulation UNN: Cooperative Move Plank

The cooperative Move Plank environment tackles a task requiring coordination to move a large object, beyond the capabilities of a single robot. In practice, this environment features two robotic arms and a large plank to be moved from its support to another location (see Figure 4.21). Two target points are defined on the plank for each end-effector. The UNN uses the process described in Figure 4.16. Formally, we have:

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

- Reward:

$$R_t = \begin{cases} -\alpha \times (D_1 + D_2) & \text{if} \sum_{i=1}^{2} c_i = 0 \\ 0 & \text{if} \sum_{i=1}^{2} c_i = 1 \\ \frac{\beta}{D_t} & \sum_{i=1}^{2} c_i = 2 \end{cases} \tag{4.7}$$

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

- Action space: In this case, the *vanilla* and UNN output for each agent is enhanced with an offset vector for the global position of the flying mobile base. Thus, for the UNN case, the action space is as follows: $a^{\mathscr{T}} \in \mathbb{R}^{(3+3)\times 2}$.

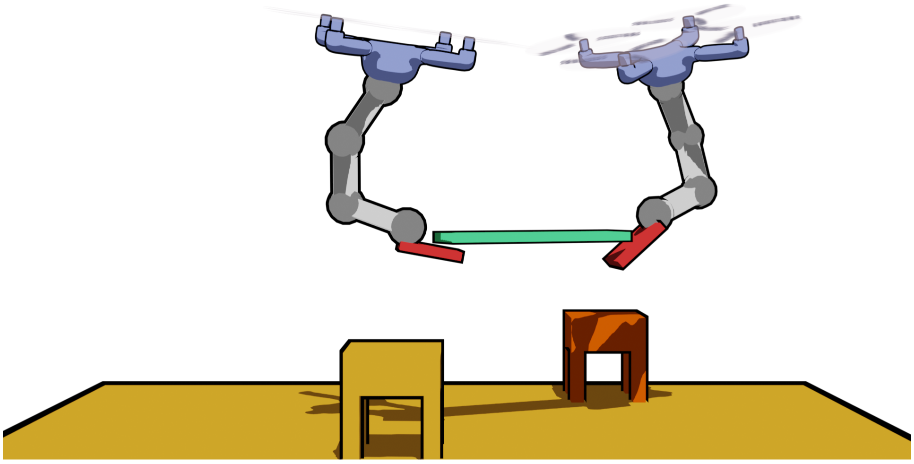


FIG. 4.21. Cooperative Move Plank Task.

Figure 4.22 displays the mean reward evolution along training for the original G3-G3 configuration as well as the G3-G2 transferred configuration. For this environment, despite a slower initial progression, the *vanilla* PPO agent is able to reach slightly higher mean episode reward than the UNN-based agent. The transfer phase however proves unsuccessful for the non-UNN process, as the new PPO agent, although able to move towards the target point, completely fails to lift the object. In contrast, the transfer UNN-based agent starts with a mean episode reward superior to all other models and requires only 30% of the initial training time to master the task as the original UNN agent, with respect to the mean reward per episode.

Once the agents and their transfer counterparts are trained, we enter the testing phase. To evaluate the performances of the various configurations, we define the best agent as the one capable of reaching to the plank and keeping it in the proximity of the goal position for the longest time. Again, we average this metric expressed in timesteps over 100 episodes with a time limit of 2000 timesteps. The results in Table 4.3 were created from these tests, and although the *vanilla* methodology is competitive in the initial training phase,

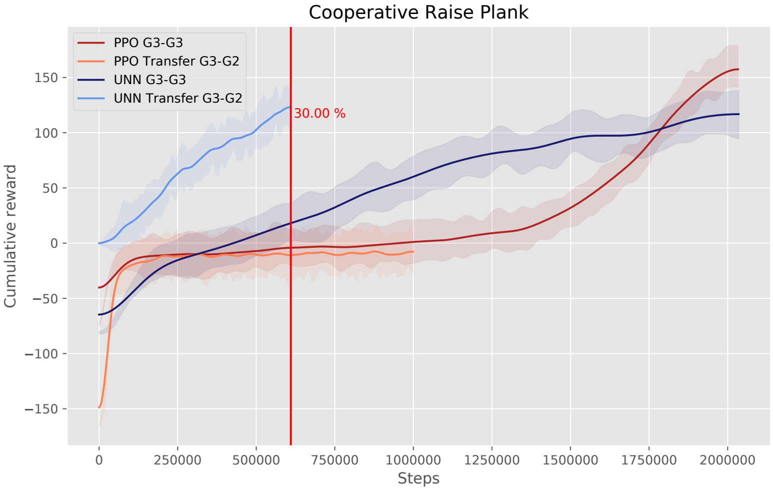


FIG. 4.22. Cooperative Move Plank Task results.

TABLE 4.3. Cooperative Move Plank.

		Config	Training	Performances
				(in timesteps)
		Cooperative Raise Plank		
UNN	Baseline	Generic 3 - Generic 2	100 %	510.54
	Original	Generic 3 - Generic 3	100%	552.68
	Transfer to	Generic 3-Generic 2	0%	111.20
	Transfer to	Generic 3-Generic 2	30%	547.88
PPO	Baseline	Generic 3 - Generic 2	100 %	573.10
	Original	Generic 3-Generic 3	100%	589.96
	Transfer to	Generic 3-Generic 2	0%	3.57
	Transfer to	Generic 3-Generic 2	100%	8.08

the UNN clearly confirms its relevance from a transfer perspective.

4.2.8 Cooperative BAM Manipulation: Dual Move Plank

A variant of the previous environment (Cooperative Move Plank) is the Dual-Arm setting, see Figure 4.23. While similar in concept, it presents the following differences: both arms belong to a single body and each arm is equipped with a suction pad. Consequently, we enhance the state $s^{\mathscr{T}}$ with a contact flag for each suction pad and revert to the initial action definition without the global offset for each agent.

In the Dual-Arm Raise Plank task, we compare the *vanilla* PPO approach with both the classic UNN and the BAM version. In this configuration, the BAM policy controls the heuristic version of a two-armed manipulator, as depicted on the right-hand side in Figure 4.24, while the classic UNN and the *vanilla* PPO polices are trained in an environment with two Kuka arms and transferred to two Berkeley Blue Robots. Learning progresses of the BAM, the classic UNN, *vanilla* PPO and their transferred counterparts

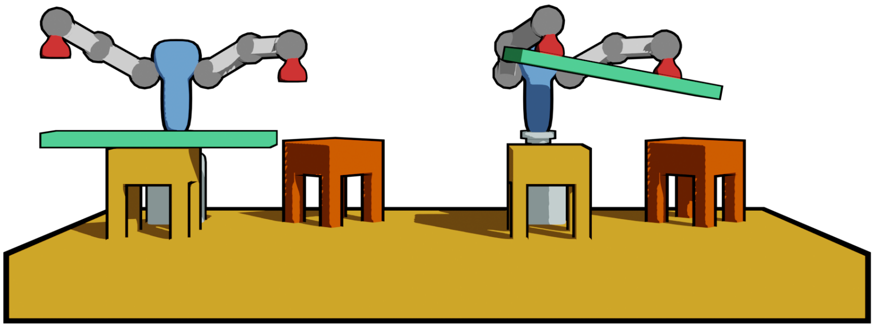


FIG. 4.23. Dual-Arm Move Plank Task.

are shown in Figure 4.25.

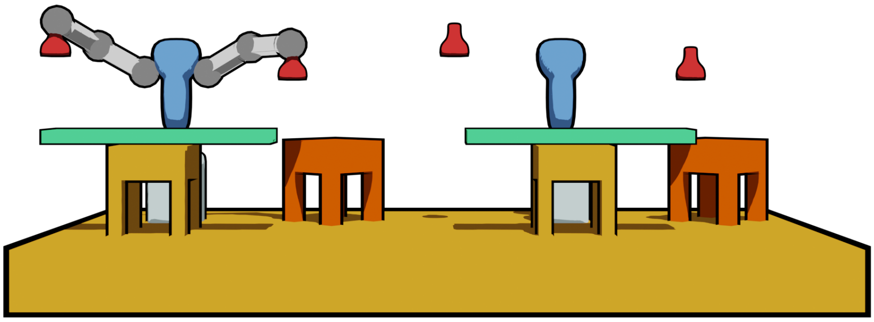


FIG. 4.24. Dual-Arm Move Plank Task.

Regarding first the classic UNN and the *vanilla* PPO, we observe that similarly to the Tennis environment, those curves indicate that even though the UNN policy starts with a slightly inferior mean reward than the *vanilla* PPO policy, the UNN policy improvement is faster and achieves a much higher mean reward. In particular, neither *vanilla* PPO policy nor its transferred version to a dual Berkeley Blue robot setup are able to succeed in this task. In contrast, the transferred version of the UNN, again controlling two Berkeley Blue robots, even though starting with a low mean reward level, quickly recovers the final performance level. While yielding a non-negligible improvement compared to the *vanilla* approach, the UNN method in this environment cannot match the efficiency of the BAM framework. Indeed, as can be seen in Figure 4.25, the BAM agent mean reward evolution presents a steeper evolution and is able to achieve two times the classic UNN mean reward,

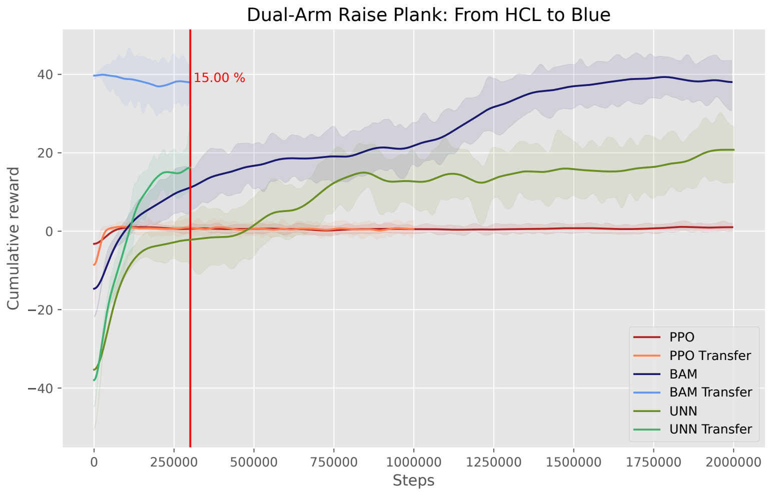


FIG. 4.25. Learning performances in Dual-Arm Raise Plank environment for PPO and UNN policies using two Kuka arms and then transferred to two Berkeley Blue robots.

thus also surpassing the *vanilla* approach. Moreover, as for transfer efficiency, it can be seen that the transferred agent immediately starts with a reward level comparable to the one achieved by the BAM policy after 2 millions timesteps, effectively showing the transfer efficiency. In this case, this indicates that the transferred policy didn't even need retraining.

As for previous environments, final efficiency is evaluated in dedicated test conditions. Specifically, in this phase, a counter is incremented for each timestep if the plank is in the proximity of a target position.

Table 4.4 confirms the results shown in Figure 4.25. As a matter of fact, the results obtained by the two UNN configurations, both classic and BAM, are clearly superior to the ones the *vanilla* policy is able to achieve. In this specific case, the BAM policy doesn't even need retraining and the transferred policy achieves a better test score than every other configuration, with the exception of the original BAM policy. The UNN policy also outperforms the *vanilla* PPO policy, even though transfer is not as efficient as in the BAM case.

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

- In the Biped Walking task:

TABLE 4.4. Dual Arm Raise Plank Summary.

		Config.	**Training**	**Performances** (in timesteps)
BAM	Original	BAM	100%	1557.05
	Transfer to	B.Blue-B.Blue	0%	1545.65
	Transfer to	B.Blue-B.Blue	15%	1547.02
	Transfer to	Kuka-B.Blue	0%	1451.03
UNN	Original	Kuka-Kuka	100%	987.11
	Transfer to	B.Blue-B.Blue	0%	4.41
	Transfer to	B.Blue-B.Blue	15%	926.66
	Transfer to	Kuka-B.Blue	15%	904.31
PPO	PPO	Kuka-Kuka	100%	139.40
	Transfer to	B.Blue-B.Blue	0%	8.45
	Transfer to	B.Blue-B.Blue	100%	152.54
	Transfer to	Kuka-B.Blue	100%	147.78

  - the qualitative appeal of the walk cycle depends on the leg configuration and not only on foot placement,
  - and from a modelling point of view, it is unclear how to design a BAM version of the task.

- While the Cooperative Raise Plank task as designed:
  - isn't affected by the arms configuration, as long as there is no self-collision,
  - but does require non-negligible coordination regarding the effectors control. Thus, unless special care is dedicated to the module training, it is likely that it might hurt the UNN policy learning performances.
  - Finally, the modelling process and constraints setup for BAM training are particularly straightforward.

These points indicate that creating a BAM version of the environment is a valuable alternative. As seen in Figure 4.25 and Table 4.4, the BAM version of the UNN behaves in a more desirable way than its counterparts, thus offering a robot-agnostic model of the task to various robots. Then, if the modules of a particular robot are sufficiently precise, it is possible to succeed in the task with no retraining at all and still yields higher performances than other policies.

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

4.3 Conclusion on TAsk-Centered methods

In this section, we presented a broad range of applications for the proposed Universal Notice Network (UNN) approach, a method that disassociates the agent control logic from the task it is supposed to accomplish. To motivate our approach, we began by demonstrating that *vanilla* transfer in RL setting is detrimental to the agent by exchanging layers between two similar agents both trained on a unique task. This experiment enlightened the fact that uncontrolled backpropagation does not structure knowledge and thus makes it difficult to forsee which model part can be shared or withdrawn without hindering performances. Consequently, the UNN approach has been applied to various tasks. For each detailed environment, we compared our technique with a classical state-of-the-art algorithm and we demonstrated the benefits and advantages of our method, both in terms of learning speed and final performance and then tackle the issue of transfer learning. Through various examples, we show that our method enables a more efficient transfer than current methods, allowing an agent to perform as well as the original policy with little to no retraining, hinting toward an important training time reduction. Additionally, the experiments set up allowed us to detect that the classic UNN architecture may convey a bias from the modules pre-training which can in various situations be used as an advantage. For configurations in which the bias can be detrimental or simply undesired, we also proposed an experimental framework, the Base Abstracted Modelling, for overcoming such biases. Despite its numerous advantages, the UNN method is in practice hindered by the feature that constitutes its main appeal. Indeed, the initial segmentation process relies, in its current form, on the hand-crafted features that constitute the interface between the task model and the control parameters (the output base). As a result, this forces each agent to adopt similar conventions to use the pretrained task model. While this does not necessarily harm performance, it heavily limits the method flexibility and range of application and although there are no theoretical constraints in building another vector interface, it has proved difficult so far to formalize the definition of such a vector. Additionally, these constraints enforce the sequential aspect of the UNN training. While this feature enables a simple form of knowledge segmentation, it is also the most likely conveyor of base biases. As a result, an alternative path worth exploring would feature a joint knowledge construction by multiple agents with different morphologies, ultimately yielding a consensus notice that would not necessarily rest on hand-crafted vectors. In the next section, we will introduced TEacher-Centered techniques that circumvent this limitation.

Publications

- **Universal Notice Network: Transferable Knowledge Among Agents** - International Conference on Control, Decision and Information Technologies (CoDIT) 2019 - Paris, France
- **BAM! Base Abstracted Modeling with Universal Notice Network: Fast Skill Transfer Between Mobile Manipulators** - International Conference on Control, Decision and Information Technologies (CoDIT) 2020 - Prague, Czech Republic
- **Universal Notice Networks: Transferring learned skills through a broad panel of applications** - Submitted to: IEEE Transactions on Systems, Man and Cybernetics: Systems

Additional Material

- https://drive.google.com/drive/folders/1rhwldaISnmpmG8bksTudQnhKvgn0Gy83?usp=sharing

Chapter 5

Teacher-Centered Transfer

Contents

In this chapter, we present two transfer learning approaches based on TEacher-Centered concepts. Indeed, as opposed to Task-Centered approaches that rely on the creation of an instance of knowledge that can be passed between agents of different morphologies, TEacher-Centered methods are designed to provide a way of distilling the knowledge of a teacher agent within students that are potentially differently structured. Building on this concept, these techniques are first thoroughly explained and their relevance is then assessed on a custom environment.

5.1 CoachGAN

5.1.1 From Task to Teacher

The previous chapter introducing the UNN underlines numerous times the critical importance of the interface between the UNN module and the agent's inherent bases. Due to the restrictions brought by this bottleneck, it becomes conceivably more complex to adapt a pre-trained UNN to an agent that would not present the elements fitting within the interface, thus limiting the universality of this method, as shown in Figure 5.1. In an effort to overcome this potentially hindering issue, projecting the UNN module output in an intermediate latent space appears as an interesting candidate. This idea would be able to provide values corresponding to measurements instead of direct orders.

Let us consider the experimental environment Dual-Arm Manipulation, introduced in the previous chapter: in this case, instead of returning translation vectors and relative effector positions for a full episode and using the reward function to backup and evaluate the actions, the UNN could provide a desired state measure of the manipulated object that should then be matched by the agent arms, as displayed in Figure 5.2. In this view, the state measure could be very diverse, as long as it can be related to the agents actions through a differentiable function, to ensure that backpropagation is possible.

5.1.2 The CoachGAN method

When assisting to a sports game, even non-expert individuals are able to detect particularly efficient players. After watching enough, observers are usually able to provide a high-level description of an ideal performer and additionally may be capable to distinguish an expert's action from a novice's. The CoachGAN method is a loose transposition of this situtation in the GAN framework where the external observer is a discriminator and players are generators.

Building on the concept of expert/teacher/student, where a teacher can use observations made during the expert class to guide its students although he was not the initial transfer target, CoachGAN relies on a two-step adversarial process to train a student agent by using expert knowledge via a common teacher critic. Specifically, the main idea is to repurpose a GAN discriminator (the teacher) trained on a set of expert trajectories to provide an error signal for a student trying to accomplish the same task. As the very

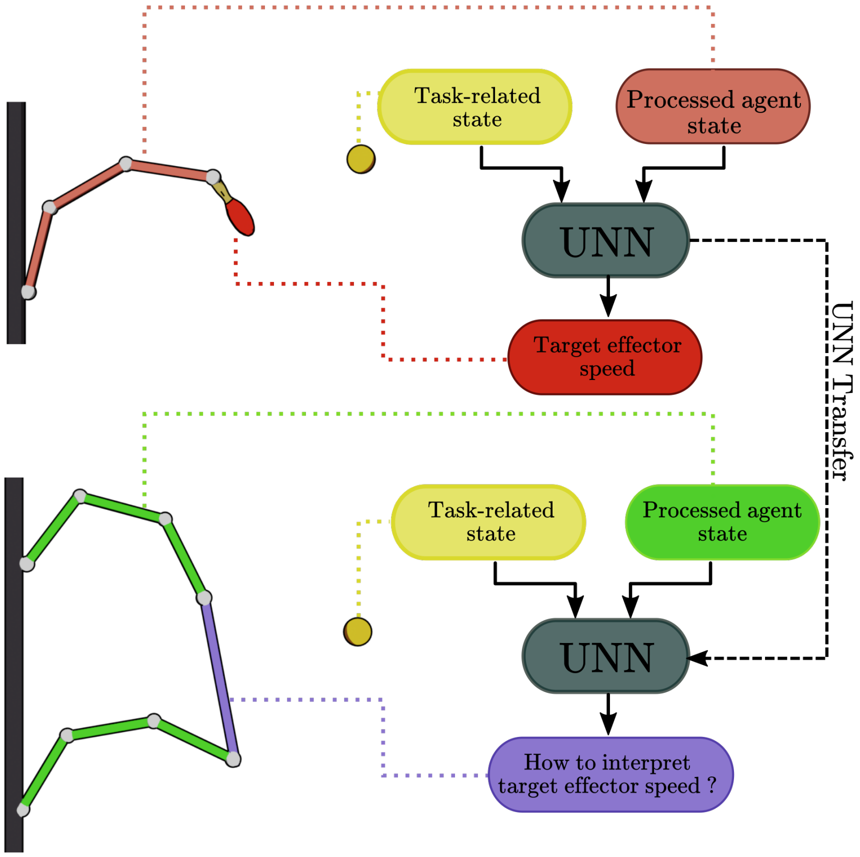


FIG. 5.1. In its current formulation, the UNN framework is likely to find limitations for cases where the UNN output instruction is not directly applicable to the transfer target.

essence of this work is to focus on differently shaped entities, it is crucial that the student and the teacher can share a common understanding for the student to use the discriminator evaluation to backpropagate accordingly its error. Consequently, this method also introduces ISV (Intermediate State Variables), an innovative yet simple way to translate the discriminator's value into a learning signal.

As opposed to most GANs applications where the generator is the only desired model and where the discriminator is generally discarded, the CoachGAN method uses the discriminator to permit the transfer to the student. The main concept of this approach is to define an ISV that can be evaluated by the (teacher) discriminator, based on representations learned on the expert trajectories, and to which the student can relate. It will then be possible to backpropagate the student error within its model and optimize it based on the teacher discriminator evaluation. The ISV is critical to the CoachGAN approach. Indeed, as the expert and the student have different morphologies, it is not straightforward how to directly compare their states/actions. The ISV role is to bridge the gap between the two agents by being the representation of a common, user-defined, intermediate state.

It would have been possible to guide the student by using the distance between the student and the expert's ISV as a loss value. However, while theoretically viable, this option is practically more complex to set up due to the discriminator pre-convergence. Specifically,

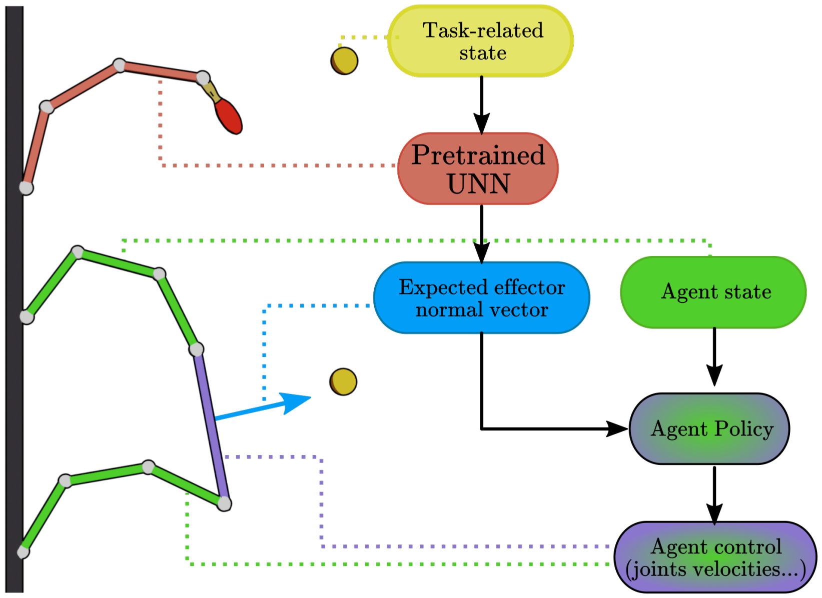

Task-related state
Pretrained UNN
Expected effector normal vector
Agent state
Agent Policy
Agent control (joints velocities...)

FIG. 5.2. It is possible to extend the UNN approach by implementing a less rigid interface between the task model and the agent.

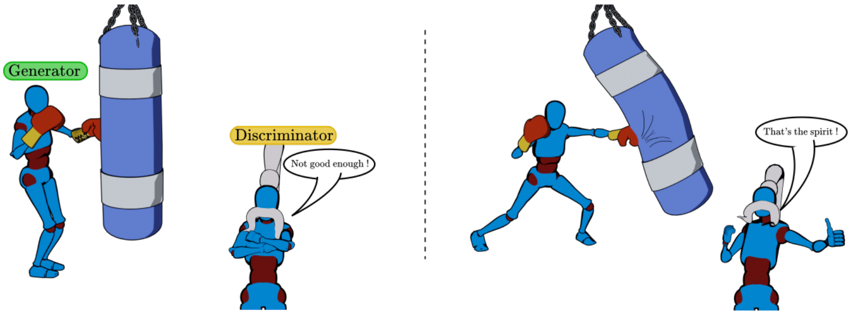


FIG. 5.3. An educated enough external observer is able to provide a feedback on the actions performed, which can be used, to some extend, to improve performance. In this stylized example, the discriminator evaluates the fighter ability through the heavy bag movement.

in adversarial frameworks, the discriminator often trains faster than the generator, which, in extreme cases, can even prevent the generator from learning as all its samples are rejected by the discriminator. This constitutes an important drawback in most use cases and may requires extensive hyperparameter tuning. For this reason, it is more straightforward to rely on the teacher discriminator to train the student generator.

The first stage aims at training the teacher discriminator to create a relevant task critic. Using Equation 2.45, the objective becomes:

$$\begin{aligned}\min_{G_T}\max_{D_T} \mathbb{E}_{x_{\mathrm{ISV}}\sim\mathbb{P}_T}[\log(D_T(x_{\mathrm{ISV}}))]+\\ \mathbb{E}_{\tilde{x}_{\mathrm{ISV}}\sim\mathbb{P}_g}[\log(1-D_T(\tilde{x}_{\mathrm{ISV}}))]\end{aligned} \tag{5.1}$$

where G_T, D_T are respectively the fake expert generator and teacher discriminator, $x_{\mathrm{ISV}} \sim$

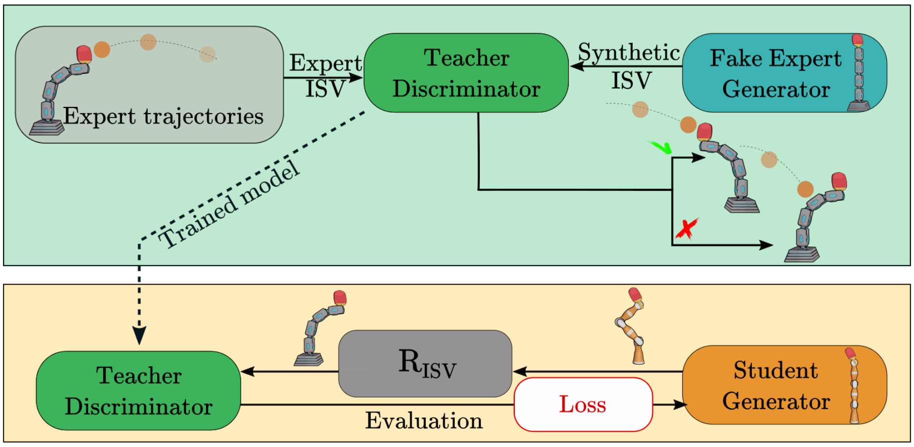


FIG. 5.4. The CoachGAN principle. The green frame depicts the teacher discriminator training process, that is then used for training the student generator (orange frame).

$\mathbb{P}_T$ represents ISV vectors from the expert dataset and $\tilde{x}_{\text{ISV}} = G_T(z)$ are synthetic ISV vectors from the fake expert generator, which sole purpose is to train the teacher discriminator. Once trained on the expert trajectories, the discriminator, when presented with an ISV, will return a scalar value ranging from 0 to 1, indicating how likely it is that this sample was generated by the expert.

Then, for the transfer, a new generator relevant to the student control dimensionality is created and paired with the teacher discriminator. In this configuration, the student loss is:

$$L_S = ||D_T(R_{\text{ISV}}(G_S(z))) - 1||^2 \tag{5.2}$$

Equation 5.2 states that the student generator aims at maximizing the discriminator evaluation given to a synthetic ISV-translated student generator solution. Practically, for a given input z, the student generator solution $G_S(z)$ is passed to the differentiable operator R_{ISV} that computes a discriminator-understandable ISV vector. Given that every function in this chain is differentiable, it is consequently possible to backpropagate the discriminator evaluation within the student's parameters for an optimization step. Figure 5.4 shows a schematic view of the method.

5.1.3 WGAN-GP for a suitable learning signal

As introduced in Section 2.5.3, training in the adversarial framework is notoriously difficult due to the multiple failure modes (discriminator pre-convergence, generator mode collapse). However, since the initial GAN formulation (40), multiple alternative loss functions and architectures have been proposed to mitigate some of the issues mentioned above. In the CoachGAN case, as the teacher discriminator is expected to be used for the downstream transfer, it is necessary that the discriminator sample evaluation provides a suitable signal, learning-wise. Taking into account these various constraints, the WGAN-GP (Watterstein GAN with Gradient Penalty (114)) appeared as a natural architecture candidate. Indeed, the WGAN-GP design, beyond making the generator training easier, also formulates the discriminator evaluation in a way that enables it to be repurposed for downstream tasks. More specifically, as a task-related vector must be passed to the networks to provide relevant information on the current state of the world, the CoachGAN method implements a Conditional WGAN-GP, yielding the following formulation for the discriminator:

$$L_D = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(c,\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(c,x)] + \lambda P_g \tag{5.3}$$

where c is the task related vector, which corresponds to external information that will be required for any agent, such as the ball speed and position, λ is the gradient penalty weight and P_g the gradient penalty that acts as a regularization parameter for the discriminator:

$$P_g = \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}}D(c,\hat{x})|| - 1)^2] \tag{5.4}$$

where $\hat{x}$ is a vector interpolated between x and $\tilde{x}$ with an interpolation factor sampled uniformly between 0 and 1. The generator optimization relies on the following scheme:

$$L_G = -\mathbb{E}_{z\sim\mathscr{L}}D(G(z,c),c) \tag{5.5}$$

where z is a random vector sampled from the generator latent space $\mathscr{L}$.

5.1.4 Task and Intermediate State Variables

To demonstrate the usefulness of the CoachGAN approach, a 2D version of the UNN Tennis environment was created. This environment features the playing agent, a serial manipulator equipped with a bat, and a ball thrown with an initial velocity, as shown in Figure 5.6 (left box). The agent's goal is to position its effector in order to intercept the ball. As mentioned above, the CoachGAN approach relies on intermediate state variables for allowing the teacher discriminator evaluation to guide the student. One of the main strength of this approach is that these intermediate state variables can be very diverse as long as the student can relate to them through a differentiable model. To illustrate this method capabilities, three configurations for this approach are provided:

Actual Kinematics: In this first configuration, the intermediate state variable is the effector position. Specifically, for a given ball configuration, the teacher discriminator is trained to distinguish between suitable effector positions (that is, likely under a task expert policy) and synthetic position from the fake expert generator. The student generator will output the target joint angles. To evaluate the student generator proposition, a differentiable forward kinematics (FK) model is used to translate the joint angles to the effector position that can thus be assessed by the discriminator, as shown in Figure 5.5. The FK model used in this configuration has no trainable parameters and is only used to keep track of the gradients.

Approximated Kinematics: This second composition still relies on the effector position, but this time the FK model is replaced by a neural network trained on the student

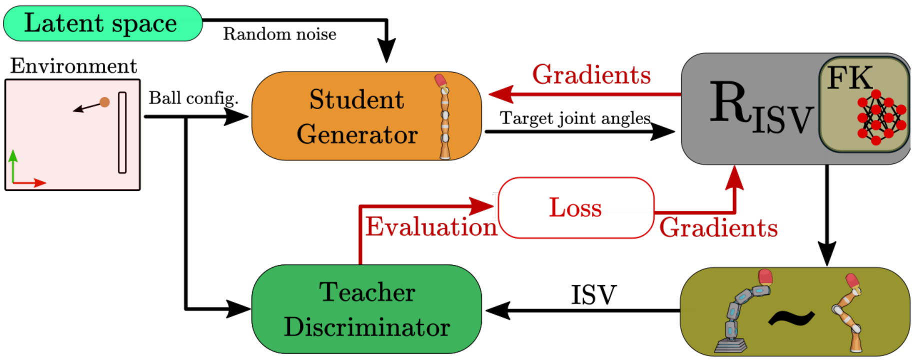


FIG. 5.5. Student Generator (in orange) optimization in experiments **Actual Kinematics** and **Approximated Kinematics**. Discriminator evaluation passes through the R_{ISV} module before reaching the generator.

configuration to compute the effector position given joint angles, also shown in Figure 5.5.

Ball Rebound Speed: The effector position is a very convenient variable when considering serial manipulators. However, it may be less relevant in cases where agents exhibit a very different structure or in other types of tasks as well. Thus, to illustrate the Coach-GAN method applicability, this third configuration demonstrates that it is possible to transfer task knowledge through measurements not directly accessible. Specifically, this case proposes to train the student using the ball velocity. This configuration requires an additional ball speed predictor model, trained to predict ball speed at the next step, given the ball configuration and the effector position, as displayed in Figure 5.6.

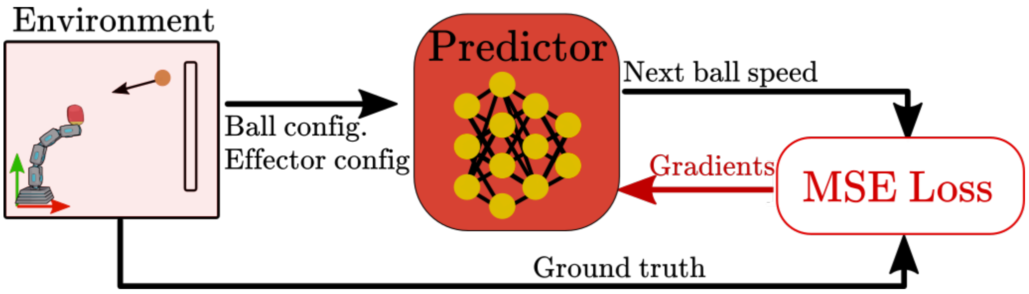


FIG. 5.6. Training predictor to forsee next step ball speed.

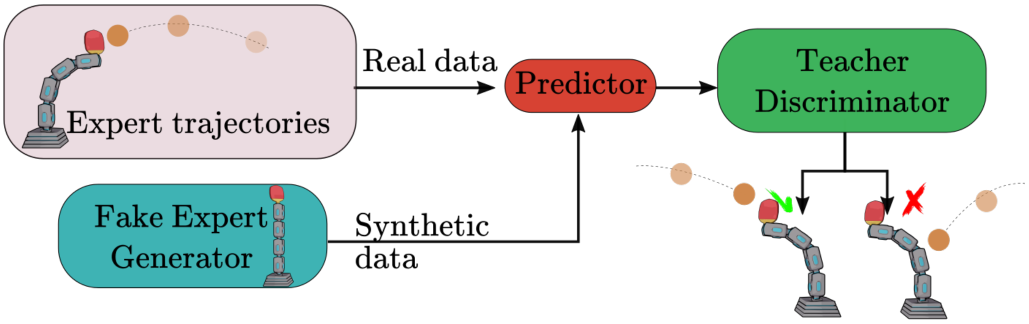


FIG. 5.7. Training discriminator based on predictor outputs.

Training the discriminator consequently requires an additional step, during which effector positions, both real and synthetic, are first passed through the predictor. This way, the teacher discriminator learns to classify suitable ball speeds, based on the predictor predictions, see Figure 5.7.

Once the teacher discriminator is trained, it is used to train the student generator. As done precedently, the student generator proposes angles resulting in an effector position, computed either with an analytical or approximated model, which is then used by the predictor to compute the ball speed at next step. The discriminator evaluation of this vector is finally used to optimize the student generator's weights, see Figure 5.8

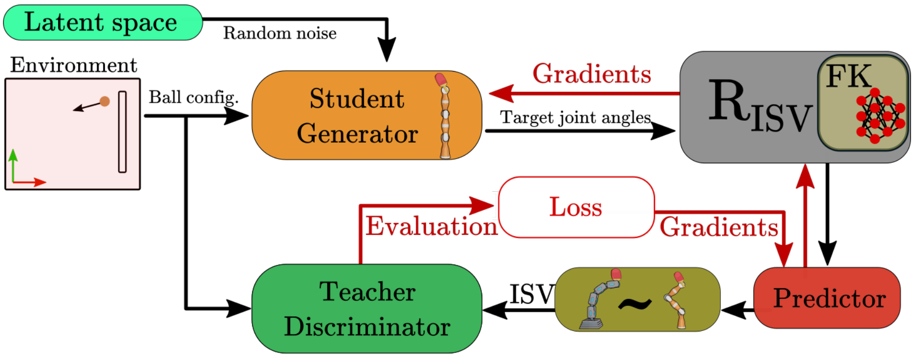


FIG. 5.8. Student generator (in orange) training setup for experiment **Ball Rebound**. Discriminator evaluation goes through the predictor and R_{ISV} before modifying generator parameters.

5.1.5 Training the Expert

In the experiments presented below, a task expert is required. This expert agent is a 4-DoF serial manipulator, trained using PPO. Consequently, a MDP version of the Tennis environment is defined and encourages the agent to maximize the following reward function:

$$r_{\text{tennis}} = \alpha + o \times (\beta + \gamma * \exp(-d)) \tag{5.6}$$

where α is a small constant that incites the agent to keep the ball over a height threshold for as long as possible. o is the binary contact flag: 1 when a contact between the ball and the wall is detected, and going back to 0 at the next step. β and γ are constant values used to weight the relative importance of accuracy when touching the wall. Finally, d is the vertical offset between the ball and the target on the wall. This reward function incites the agent to hit to ball towards various locations which broadens the ball impact distribution, thus improving the agent versatility and ultimately providing a more diverse trajectory dataset.

5.1.6 CoachGAN Results

5.1.6.1 Effector Position ISV

Once the expert agent is ready, it is used to generate the training dataset. For the two first configurations, that is **Actual Kinematics** and **Approximated Kinematics**, the proposed ISV is the effector position. As such, each line of the gathered dataset features the initial ball conditions (position and speed) and the expert effector position recorded when contact is detected.

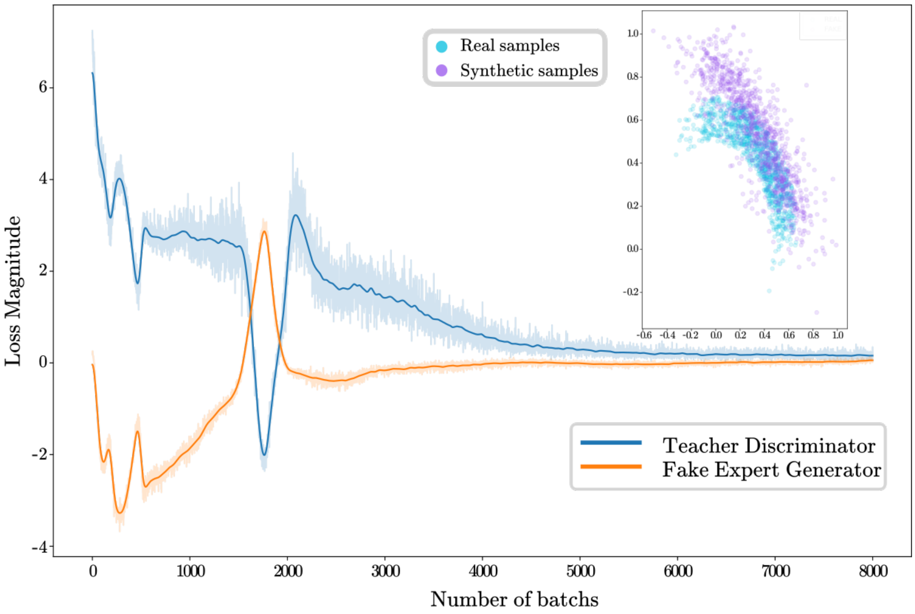


FIG. 5.9. Adversarial losses along training.

The adversarial pair (teacher discriminator and fake expert generator) is then trained on this trajectory dataset. Figure 5.9 shows the adversarial losses along training based on Equations 5.3 and 5.5. These alternating curves are usual in adversarial framework and represent one model improving in spite of the other. It is however possible to see that the process eventually reaches an equilibrium. As a visual In the upper right corner are displayed fake expert generator ISV solutions in the task space for various ball configurations, showing that the real expert behaviour was indeed approached.

Figure 5.10 displays the normalized discriminator evaluation of ISV (effector positions in this case), for a given ball configuration. As can be observed, the discriminator presents a clear preference for ISV along a line following the expected ball path. Furthermore, the discriminator places its highest confidence (strong yellow color) in an area located in the close vicinity of the dataset ground truth (the expert effector position, drawn in green), showing that it has indeed learned from the expert behaviour.

Once trained, the discriminator can be repurposed for training the transfer target, which is the student generator. In this configuration, the task knowledge is transferred towards a 5-DoF serial manipulator. As such, the generator output vector is interpreted as the target joint angles for the student. The effector angle is manually set to $\frac{\pi}{4}$ degrees. Then,

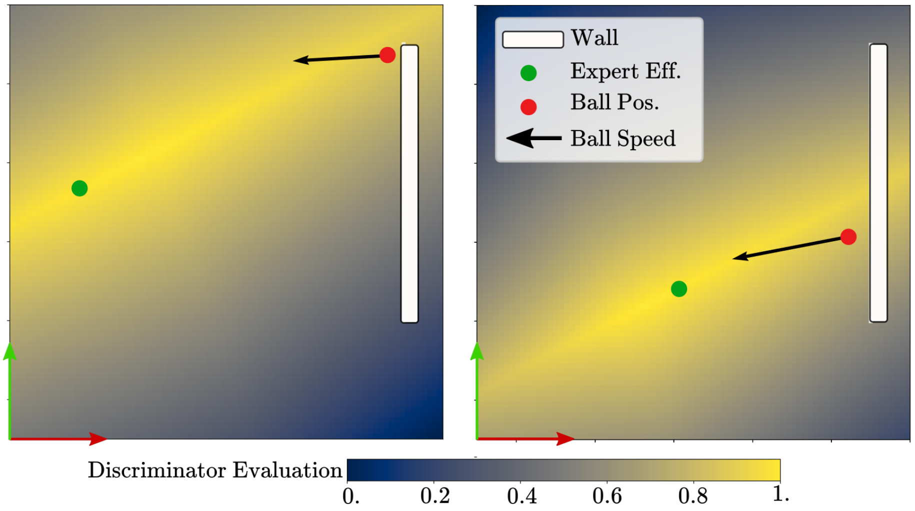
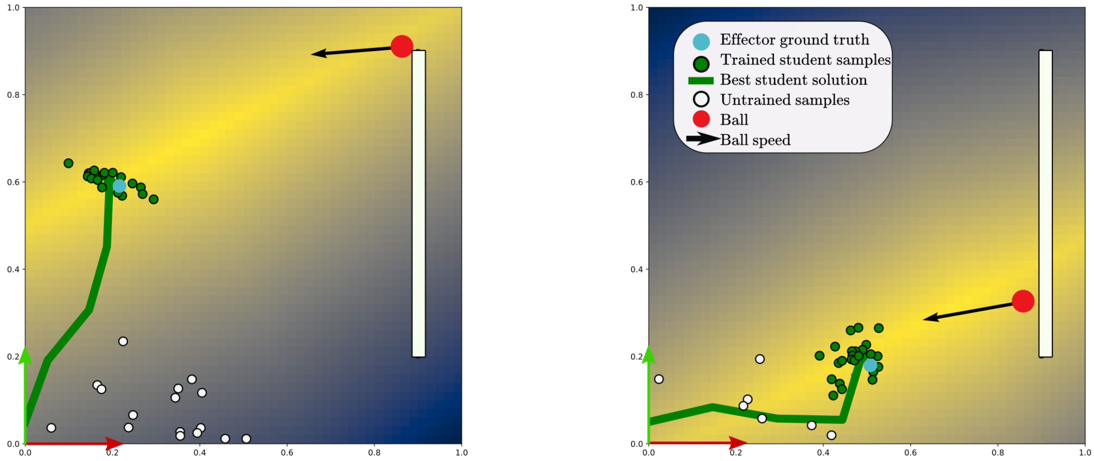


FIG. 5.10. Normalized discriminator evaluation of ISV (effector positions) in the task space for two given starting ball configurations. Higher values (yellow) are favored.

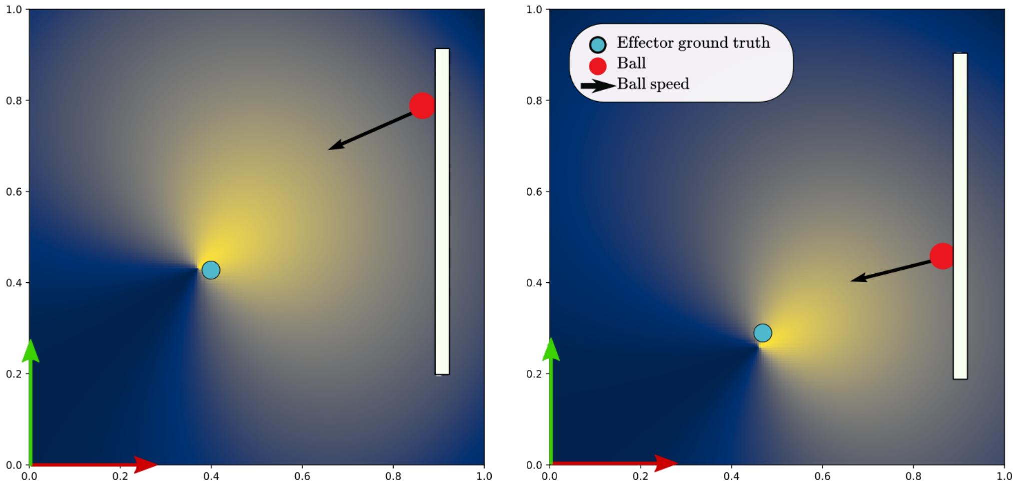


(a) Actual FK.

(b) Approximated model.

FIG. 5.11. Sampled solutions for actual and approximated ISV in green. Baseline from untrained generator in white.

it is necessary to translate the student generator output into the expected intermediate variables, that is the effector position, with the help of the unparametrized model. Figure 5.11a shows, in green, various ISV solutions given by the student generator, after training. The best solution, with respect to the discriminator, is also fully displayed. As a baseline, solutions from an untrained student are also displayed in white. As can be seen, the trained solution distribution is more compact and centered in the area most valuated by the teacher discriminator, as opposed to the untrained distribution. This demonstrates that it is possible to guide a generator with a discriminator through intermediate variables. Furthermore, it is possible to notice that the trained generator solution results in a student effector close to the original expert solution (light blue).



(a) Configuration A.

(b) Configuration B.

FIG. 5.12. Task-space evaluation by the discriminator based on predictor results.

5.1.6.2 Approximated Kinematics ISV

Focus is now set on the **Approximated Kinematics** configuration. In this case, an approximation model is trained to predict the student effector position given the agent joint angles. Using the same teacher discriminator model as in the **Actual Kinematics** stage, the student generator is trained similarly, replacing this time the agent analytical kinematic model by the trained one. Similar to the previous case, Figure 5.11b shows trained generator solutions and their baseline. As can be observed, the new generator solution distribution is also compact and adequately follows the discriminator preferences, demonstrating the reliability of the approximated model. While the approximated model is simple, this approach demonstrates that the CoachGAN is not limited to analytically defined models and can be used for cases where no robot kinematic model is available and must then be learned.

5.1.6.3 Ball Rebound Speed ISV

Finally, let us consider the last configuration where, instead of relying on the effector position, the discriminator uses the **Ball Rebound Speed** to distinguish between expert trajectories and synthetic/not suitable ones. Figure 5.12 displays the discriminator evaluation of effector positions based on the predictor outputs as introduced in Section 5.1.4. Due to the fact that the teacher discriminator training involves the predictor, the trained discriminator preferences do not overlap the distributions observed in Figure 5.10. However, they still strongly favor positions located closely to the expert ground truth. The student generator is then trained accordingly to the process depicted in Figure 5.8.

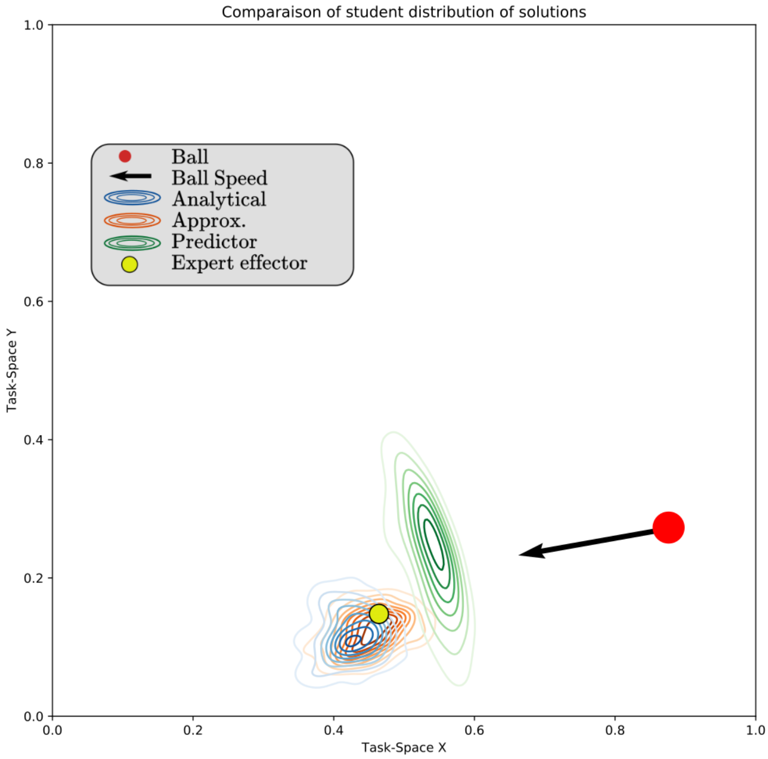


FIG. 5.13. 2D representation of the solution distribution. Successful solutions are either overlapping expert position or in the ball path.

5.1.6.4 Analysis

The performances of these models are displayed in Figure 5.13 where, for a given ball configuration, each model samples 2048 solutions. Sampling several predictions from the model enables us to conduct an analysis and evaluate, for instance through the width of the proposed distribution of solutions, the variance of the model. Indeed, it is possible to notice that the blue and orange distributions, corresponding respectively to the analytical model and the approximated kinematics model, are very close, showing the approximated model reliability for transferring the discriminator evaluation. Moreover, these distributions overlap the expert effector position. The green distribution, representing solutions from the generator trained with the predictor pipeline is slightly broader than the two other distributions, as well as being located closer to the ball. As training for this model relied on another dataset, it consequently generated a differently inclined discriminator, leading in turn to a closer player student. Still, it appears that these distributions are directly located in the ball speed direction, thus enhancing the agent chances of intercepting the ball. Furthermore, it is possible to explain the width of the distributions by the fact that only the effector position is registered on contact and that the model is not directly aware of the bat length. Nonetheless, as shows Figure 5.14, all generators provide educated

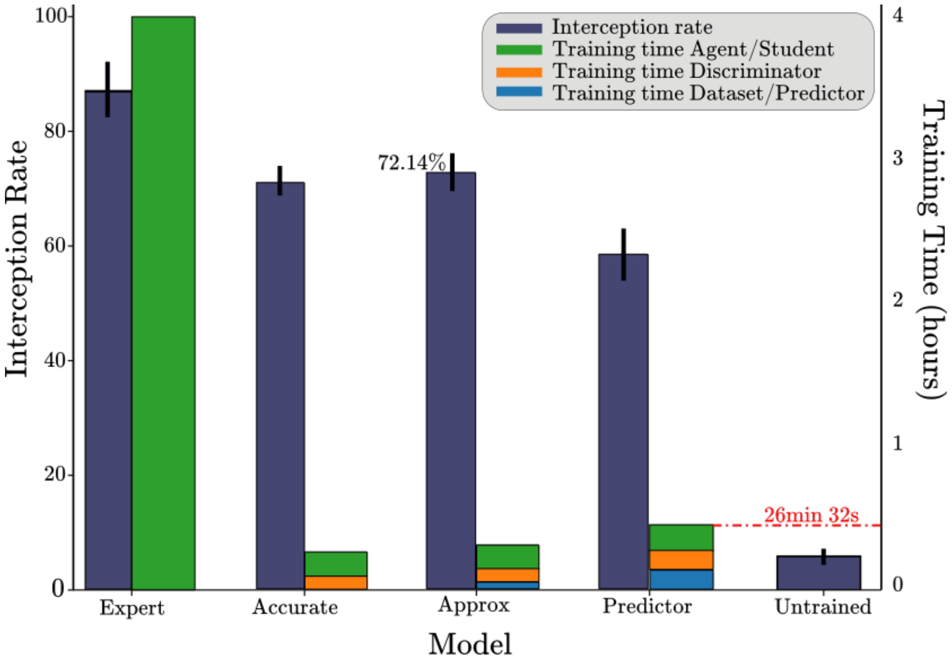


FIG. 5.14. Interception performance and training time.

solutions. To establish these results, 500 ball trajectories are recorded, and, for every case, each student samples one joint angles solution given the initial ball configuration. Then, a check for collision is run along the ball path allowing *in fine* to obtain the interception score of each model (dark blue bar in Figure 5.14).

Now, let us consider the robots involved in the transfer experiments detailed above. While it might appear that these agents are kinematically close, it is important to note that the ISV vectors could have been generated by any robot having an end effector. Indeed, the student generator never has access to the expert geometry, thus broadening the spectrum of transfer.

Lastly, training these models requires negligible time and computation resources. Indeed, while the initial expert training requires 4 hours, the most time-consuming configuration, featuring additional dataset and predictor, needs less than 12% of total training time (26.51 minutes). Student training in itself can be completed in less than ten minutes. And, as was shown in the **Actual Kinematics** case, a discriminator can be reused, thus further reducing the time needed for transfer. Although tempting, increasing training time is not likely to yield considerable improvement. Instead, based on the teacher discriminator result analysis, a more promising approach for higher quality students would be to focus on creating a more qualitative teacher that would progressively reduce the area deemed suitable. In this view, for the **Ball Speed** configuration, an improved and more reliable predictor model would surely result in an improved student performance. Nevertheless, these results clearly illustrate that the CoachGAN method does allow fast transfer of task knowledge between differently shaped entities.

5.1.7 Conclusion on CoachGAN

In this section, a new step-by-step methodology for systematic and fast transfer of knowledge from one entity to a differently structured one has been proposed. Relying on an adversarial framework, the CoachGAN technique was evaluated using various intermediate state variables (ISV) and shows that it is possible to transfer task knowledge to a student agent without interaction of this agent with the task. As explained in the experiment results, this method allows to train students within minutes, while most RL tasks require several hours of training. Thus, the applications of CoachGAN are numerous, especially given the increasing availability of approximation models. In its current form, this method does however present some limitations. Specifically, as can be seen in the experiments, the student generator outputs correspond to final target states (for instance, the expected configuration for touching the ball). This output definition can in some cases be detrimental to the application spectrum, particularly in cases where unexpected events must be handled.

This limitation stems directly from the learning process. Indeed, as the learning loss is established based on the ISV, the discriminator learns to evaluate states. As the student generator is trained using the discriminator, predicting intermediate state conditioned on current state makes training more difficult.

Taking into account this limitation, various alternative ways could offer a way out. A straightforward approach would be to adapt the generator to output several intermediate key states and sum the teacher discriminator evaluations of these states to compute the error. Although appealing, this formulation could introduce stabilization issues related to the number of intermediate steps and the distance between them. Another paradigm would be to find a way to evaluate the results of a student action with respect to the action taken by the teacher for a similar environment configuration. The next section introduces the Task-Specific Loss (TSL), another TEacher-Centered method that relies partly on this second approach to propose a reactive agent formulation.

5.2 Task-Specific Loss

As seen in the previous section, the CoachGAN methods provides a way to transfer knowledge from a given expert to a morphologically distinct student. However, this technique relies on an architecture that prevents its usage in an interactive way. Specifically, its current form is geared toward the generation of intermediary states that must then be reached using other state-of-the-art methods.

The Task-Specific Loss (TSL) approach, introduced in this chapter, aims at tackling this issue by linking the student loss to its ISV variation.

Building on works described in (75) that introduced a Variational Auto-Encoder (VAE) architecture suited for assistive control, the Task-Specific Loss (TSL) approach introduces an extension that improves on CoachGAN interactive limitations as well as enabling the low-cost transfer of latent-spaces between differently shaped agents for the initial assistive

control method. To do so, the TSL relies on the following main idea: ideally, the closer the initial ISV state, the smaller the ISV variation between the student and the teacher should be after respective action, as shown in Figure 5.15
Assistive robotics currently have multiple applications, such as exoskeletton usage or, in a more domestic and common environment, assisting disabled people in their everyday lives. While the architecture proposed by the authors of (75) strongly ease the process of controlling dextrous robots, the TSL methodology brings an innovative way to transfer these control policies to morphogically distincts robots, thus futher easing the user experience in case switching robots is required.

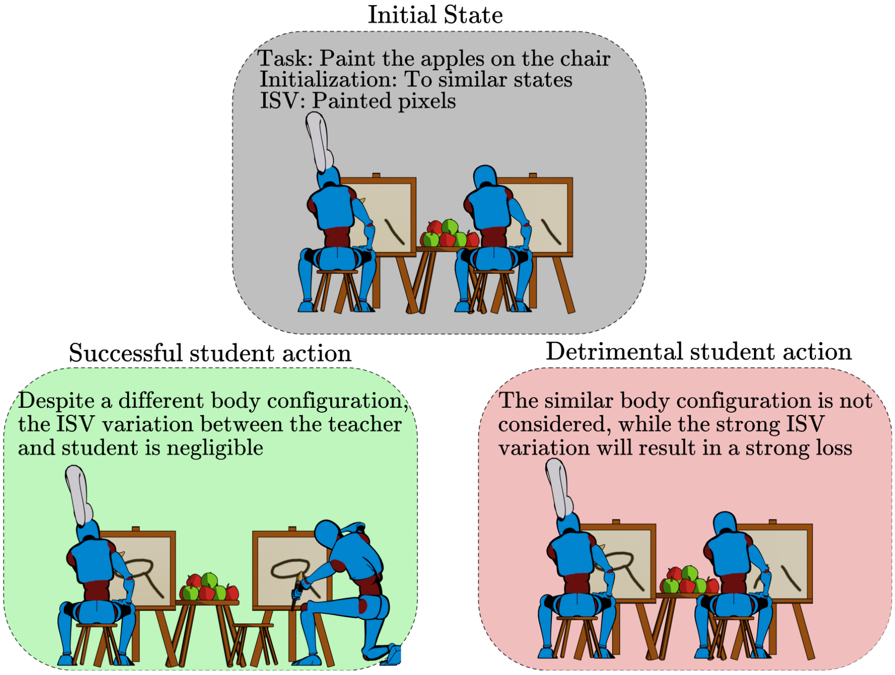

Initial State
Task: Paint the apples on the chair
Initialization: To similar states
ISV: Painted pixels
Successful student action
Despite a different body configuration, the ISV variation between the teacher and student is negligible
Detrimental student action
The similar body configuration is not considered, while the strong ISV variation will result in a strong loss

FIG. 5.15. The TSL defines an Intermediate State Variable (ISV) of interest that is responsible for sending relevant error signal to the student. In this picture, the ISV is represented by the painting. Body position having no influence on this value, the green configuration yields a low error, while the red setting returns a strong error value.

5.2.1 The TSL Principle

The TSL technique belongs to the TEacher-Centered approaches and, as in the CoachGAN process, relies on an expert dataset to transfer knowledge to the student. To do so, this instance of TEacher-Centered techniques, inspired by (75), relies on a Variational AutoEncoder to encode the expert strategy in a latent space that can later be sampled from. In the TSL framework, the efficiency of a transfer is evaluated with respect to the similarity of the variation between its ISV states and the teacher's between two timesteps. Conceptually, the TSL is supported by the belief that a trained student, placed in a given configuration, should produce an action that would lead it into a state similar to the state of the expert if it was placed in the same configuration. To reach this result, the transfer loss function focuses on the ISV variation between two steps, see Figure 5.16. The evaluation of the loss function during transfer is nevertheless more thoroughly described in Section 5.2.3

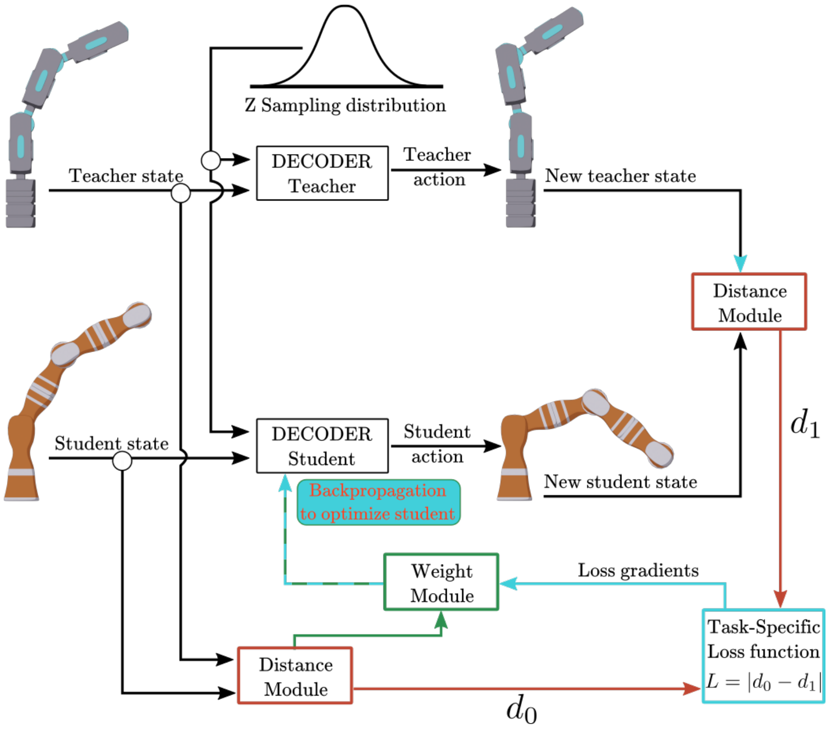


FIG. 5.16. TSL penalizes the student when its movement does not preserve the estimated metric at the previous timestep.

5.2.2 Method

Similarly to CoachGAN, the TSL is a two-step process. However, it relies on another type of generative models: Variational AutoEncoders, introduced in Section 2.6.2. Instead of relying on a discriminator to distinguish between expert-generated ISV and unlikely expert states, the TSL method leverages the encoding space that is inherent to the autoencoders class of models.

Specifically, the first step of this method is to train the VAE to be able to reconstruct the expert action, given its current state. This forces the VAE to create a latent representation of the expert trajectories, which in turn allows to bypass manually-crafted features as in the UNN framework. To fit into this new framework constraints, the usual VAE architecture has to be modified. Concretely, instead of aiming at reconstructing directly the input, the model here aims at only reconstructing the expert action. Thus, the reconstruction objective from the Equation 2.50 of common VAEs becomes:

$$L_r = \mathbb{E}[||a - \tilde{a}||] \text{ where } \tilde{a} = D(z, s) \tag{5.7}$$

where the $\mathbb{E}$ operator represents the average over a batch of examples, a is the expert

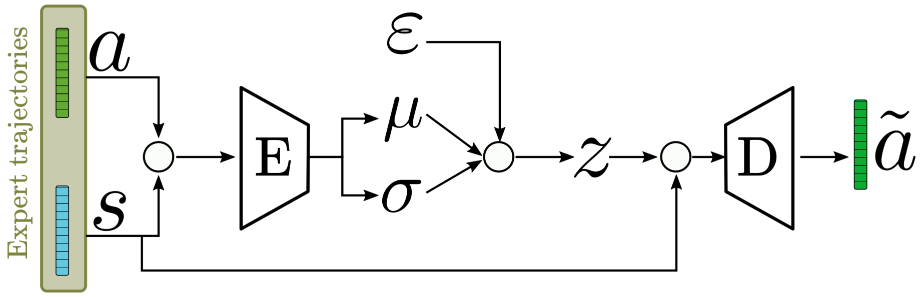


FIG. 5.17. User-assisted VAE architecture during training phase.

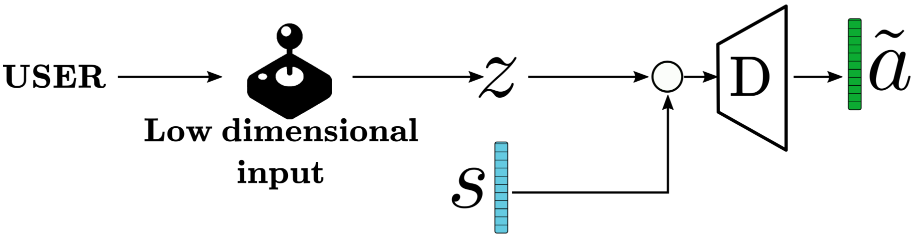


FIG. 5.18. User-assisted VAE architecture during usage/testing phase.

action and $\hat{a}$ the decoder D reconstructed version.

Conceptually, it is possible to see this setup as a way to embed the expert's policy within the *code* z. Then, it allows the decoder, conditioned by the current agent state s, to adequately reconstruct the expert input action a, even though the action dimensionality is higher than the code size, see Figure 5.17. An additional difference of the TSL framework with the previously introduced CoachGAN approach lies in the testing setup. Indeed, as the process described above results in an embedding of the expert policy, it is consequently possible to judiciously select actions from the resulting low-dimensional latent space to control the student in the same way that the teacher was controlled. In the assistive scope in which this method was developed, the action selection process can for instance be assigned to a human operator, as shown in Figure 5.18. Concretely, the action selection process returns a low-dimensional *code* z, which, once decoded by the agent, enables the control of the higher DoF robot.

5.2.3 Transfer

From the expert dataset, an intuitive agent (the teacher) has been created, it is represented by the decoder D_T shown in Figure 5.19. The goal is now to create a decoder D_S for a student agent with a body configuration that is distinct from that of the teacher agent, see Figure 5.19. The main motivation here is to leverage the previously trained model and thus avoid recreating a dataset of expert trajectories for the student agent. Indeed, there are numerous examples of systems where generating several instances of datasets could be costly, difficult or even plainly impossible, for instance demonstrations of human experts on a physical system. However, here lies the main obstacle to the transfer. Indeed, as the two agents do not share the same body structure, it is unclear how to force the student to mimic the teacher. The Task-Specific Loss (TSL) offers a conceptual answer to this issue by introducing a differentiable task-relative metric as the loss function. **Its goal is to induce a similar state variation, given a task-specific metric, for the student as for the teacher**.

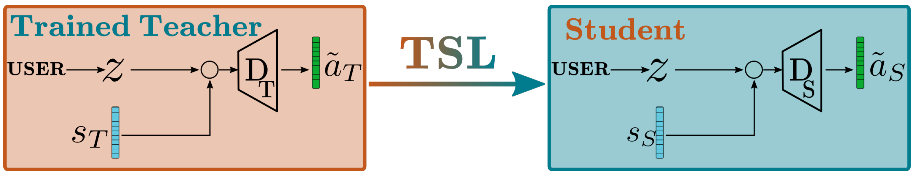


FIG. 5.19. Distillation process from the teacher to the student agent *via* TSL.

The TSL approach shares features with both Reinforcement Learning (RL) and Supervised Learning (SL), but also clearly distinguishes itself from these two approaches. Specifically, the process takes place in a Markov Decision Process (MDP), similarly to RL, but does not rely on a reward function in the classical way. Indeed, the reward signal from RL is usually delivered from an external process, thus forcing the agent's optimization to act on the action probability distribution, not on the action directly. Also, RL relies on exploration to discover suitable policies, while the TSL student agent can directly learn from the teacher. The RL exploration process is prone to lead to very low sample efficiency, whereas the TSL importantly improves this aspect, as demonstrated in Section 5.2.5.4. The TSL method also differs from Supervised Learning (SL) methods, because although it uses a loss function that recalls classical regression methods, it is not necessary to build a dataset which would defeat the purpose of the transfer from the teacher agent.

The main idea of the TSL approach is to enforce a similar state variation between both agents. This state variation is the result of the action, which is the decoder's output and is consequently differentiable. As a result, if the state modification function can be written in a differentiable fashion, it is then possible to compute directly how accurate was the student's action with respect to the teacher's. This, in turn, allows to compute an error related to the result of this action and apply the backpropagation directly to the student's weight matrices. Specifically, given a starting position for the teacher and the student agents, we first compute the value d_0, corresponding to the initial distance based on the current TSL metric, a task-defined state similarity measure. Next, a random code vector is sampled from a Gaussian distribution centered on the origin and with a standard deviation of one, in order to respect the teacher's *code* distribution induced by the KL constraint. This code z is used by both agents, along with their current state to produce an action. We then use this action to increment the state of both agents. From here, it is possible to compute the next metric value d_1, which finally leads to the TSL loss: $L_{\text{TSL}} = |d_0 - d_1|$. As both distances were obtained using differentiable functions, it is possible to directly determine the gradients, based on how close the student action was from the teacher one and optimize the student parameters accordingly.

However, it is paramount that the student mimics the teacher only when their initial states are deemed close enough, task-metric-wise. Hence, before backpropagating the loss, each element is weighted accordingly to the initial distance between the agents. Formally, we use the following weighting strategy:

$$w_i = \exp(-d_{0,i} \times \alpha_{\text{task}}) \tag{5.8}$$

That is, the importance of each batch element i is exponentially proportional to its initial distance $d_{0,i}$, where α_{task} is a regularization factor used to take into account the scale of each environment. Finally, the loss is computed as the mean of each example error and we have:

$$\mathscr{L}_{\text{TSL}} = \mathop{\mathbb{E}}_{i \in [0 \ldots n]} [L_{\text{TLS},i}] = \mathop{\mathbb{E}}_{i \in [0 \ldots n]} [|d_{1,i} - d_{0,i}| \times w_i] \tag{5.9}$$

This process is depicted in Figure 5.16.

5.2.4 Experimental setup

As introduced in Section 5.2.2, the TSL architecture is particularly well suited to enable assistive control of robots, due to the embedding of expert policies in low-dimensional latent spaces. Thus, this section introduces two experimental assistive control tasks, used to validate the proposed approach. Each of them uses two specific serial manipulators controlled by a human user through the *code* z. The setup and goal of each task are now detailed.

5.2.4.1 Tasks and Agents

Circle to circle contour displacement. Inspired from (75), the TSL principle is first evaluated on a rather simple task. In this environment, the objective is to move the end effector of a serial manipulator agent between and along two quarter circles of different radii. Once the expert is trained, its latent space will be transferred to another serial manipulator with a different structure. To train the teacher, we first create an expert dataset, that consists of sequences of states (joints angles) and actions (joints speed), using an inverse kinematics based approach, see Figure 5.17. Following the works of (75), a 2 dimensional latent space for the *code* z will be used to embed the expert policy. Regarding the transfer, the task metric is defined as the distance between the teacher and the student end-effectors. Specifically, a custom batch forward kinematic module computes the end-effector position based on joints angles and segment lengths. That is, for each batch element:

$$d_{\text{circle},i} = d_{\text{Euc}}(F(l_t, q_{i,t}), F(l_s, q_{i,s})) \tag{5.10}$$

where d_{Euc} is a module that computes the Euclidean distance between two points, F is the differentiable forward kinematics module, l_t is the teacher segments length, $q_{i,t}$ the teacher joint angles at time i and we use respectively l_s and $q_{i,s}$ for the student and the weights w_i are computed using Equation 5.8. Then, for a whole batch of examples, we have:

$$\mathscr{L}_{\text{circle}} = \underset{i \in [0 \dots n]}{\mathbb{E}} [w_i \times |d_{\text{circle},i} - d_{\text{circle},i+1}|] \tag{5.11}$$

Initially learned by a 5 rotoid joints robot (5R), this task knowledge is then distilled within a 4R agent, which total length equate that of the 5R, through the TSL $\mathscr{L}_{\text{circle}}$ using this Euclidean distance between end-effectors as a metric, see Figure 5.20.

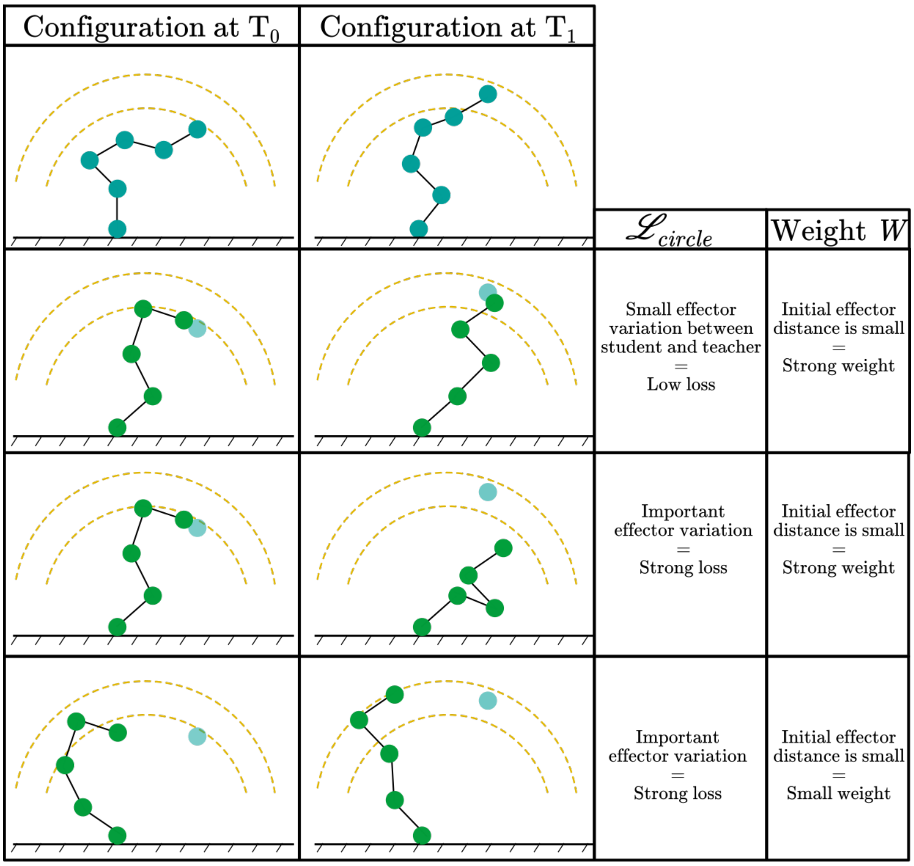

Configuration at T_0
Configuration at T_1
$\mathscr{L}_{circle}$
Weight W
Small effector variation between student and teacher = Low loss
Initial effector distance is small = Strong weight
Important effector variation = Strong loss
Initial effector distance is small = Strong weight
Important effector variation = Strong loss
Initial effector distance is small = Small weight

FIG. 5.20. Various *teaching* cases and their corresponding losses and importance weight. The teacher's joints are drawn in blue, while green is used for the student.

Tennis Task. In this second environment, designed to evaluate our method in a more complex setting, the goal is to bounce a ball against a vertical wall as many times as possible. Despite the conceptual similarities, it is distinct from the one used in Section 5.1.4 where the objective was mainly to intercept the ball, while this environment emphasizes the goal of maximizing bounces. The task is initially performed by the teacher, a 4R serial manipulator, the last segment serving as a bat to bounce the ball. The expert dataset is generated by a custom RL-agent trained on this task with an additional signal to specify the ideal impact location and a 1D latent space is used for this task. The tennis skill is then transferred to two students: a 5R and a 3R agent, with a total length equal to the 4R's. This time, the TSL relies on three terms: the Euclidean distance between end-effectors positions, similarly to the previous environment and referred to as $\mathscr{L}_{\text{circle}}$, the difference in the orientation of the bats and the difference between the magnitude of the actions (joints speeds):

$$\mathscr{L}_{\text{tennis}} = \mathscr{L}_{\text{circle}} + \mathscr{L}_{\text{ori}} + \mathop{\mathbb{E}}_{i \in [0...n]} [||A_{i,t} - A_{i,s}||] \tag{5.12}$$

where $\mathscr{L}_{\text{ori}}$ is the equivalent of $\mathscr{L}_{\text{circle}}$ for the bat orientation differences, when $A_{i,t}$ and $A_{i,s}$ are respectively the teacher and student actions at time i.

5.2.4.2 Models

The VAE model for the teacher agent is composed of an encoder and a decoder. The encoder part is composed of one 64 units hidden layer, followed by a split leading to the mean and standard deviation heads having also 64 units each. These two layers allow generating a code of smaller dimension that is then sent to the decoder. This latter model is a feedforward architecture with two hidden layers with 64 units each. Both the encoder and the decoder use Tanh non-linearities. We initialize both networks using Xavier initialization (145). For the transfer case, we remove the encoding part and train only the decoder using the TSL as explained in 5.2.3 and shown in Figure 5.19.

5.2.5 Results

5.2.5.1 Circle to Circle Contour Displacement

The teacher model has been trained using mini-batches of 4096 examples, an Adam optimizer with a learning rate of 5×10^{-3} and a weight decay value of 10^{-3}. Figure 5.21 shows the loss evolution of the training set for the teacher agent and the distribution of the latent representation z of a randomly sampled batch of 4096 examples.

The use of a VAE instead of a *vanilla* auto-encoder has been motivated by the idea that the VAE will create a smooth distribution of the latent representation, instead of having sparse clusters. It is here clearly validated, as can be seen in Figure 5.21: in both directions, the z distribution appears to be dense, without obvious holes or clusters. Furthermore, VAEs are known for their disentangling capacities, as explained in Section 2.6.2. As the expert dataset is composed of trajectories between and along two circles of different radii, we are thus able to generate a 2D controller whose main axes respectively translate into radial and axial movements.

Once the teacher is ready, it becomes possible to create a student agent that mimics the teacher behaviour on the task. For this specific task, we use the TSL described in Equation 5.11 that penalizes the student when the variation of its states, TSL metric-wise, differs from the teacher states variation. For similar starting states of the teacher and student agents and given the exact same sequence of user commands $\{z_0, z_1, ..., z_t\}$, the agents ending states should also be similar, TSL metric-wise.

The main plot in Figure 5.22 shows the resulting distance between both effectors after 10 successive actions, for a given z and a starting position in which both effectors are

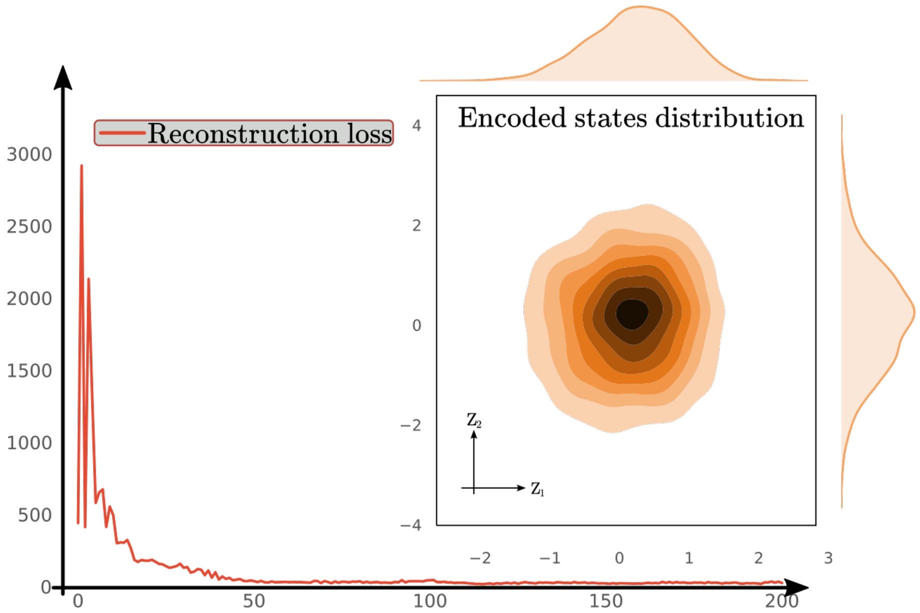


FIG. 5.21. Reconstruction loss along training epochs and 2D code distribution on the test set for the circle task.

located at the same place. It is possible to see the most movements result in a distance inferior to 1% of the total body length, an error that is completely acceptable for the tasks considered. In the worst cases, with no user adaptation, the difference is still less than 3% of the total body length, underlining the transfer efficiency.

Additionally, using the experimental conditions described for the distance distribution, we plot in Figure 5.22 a small subset of configurations, for a clearer understanding of the movement. In these cases, the student and teacher end-effectors end up in a close configuration, as enforced by the TSL, even though there was no closed-loop correction coming from the user. Even though the student has not been taught on a dedicated dataset, it has still been able to generate movements similar to the teacher's, demonstrating the transfer efficiency.

5.2.5.2 Tennis task

The Tennis task was also used as a benchmark and testing environment for the TSL. Indeed, its complexity as well as the fact that it involves physics, external perceptions and world interactions makes it appealing for evaluating the TSL.

The objective is to create an assistive agent to facilitate the control of a serial manipulator playing a game of tennis and allow the user to rely on a single latent variable to get the agent to bounce the ball. This environment is more challenging than the **circle to circle displacement** because it implies a higher reactivity from the agent. Furthermore, writing an analytical model to solve this task proves to be challenging. The first step consists in creating an expert player through RL. To fulfill this step, we used PPO once more. We recall here the main components of this environment where the teacher is a 4-DoF agent:

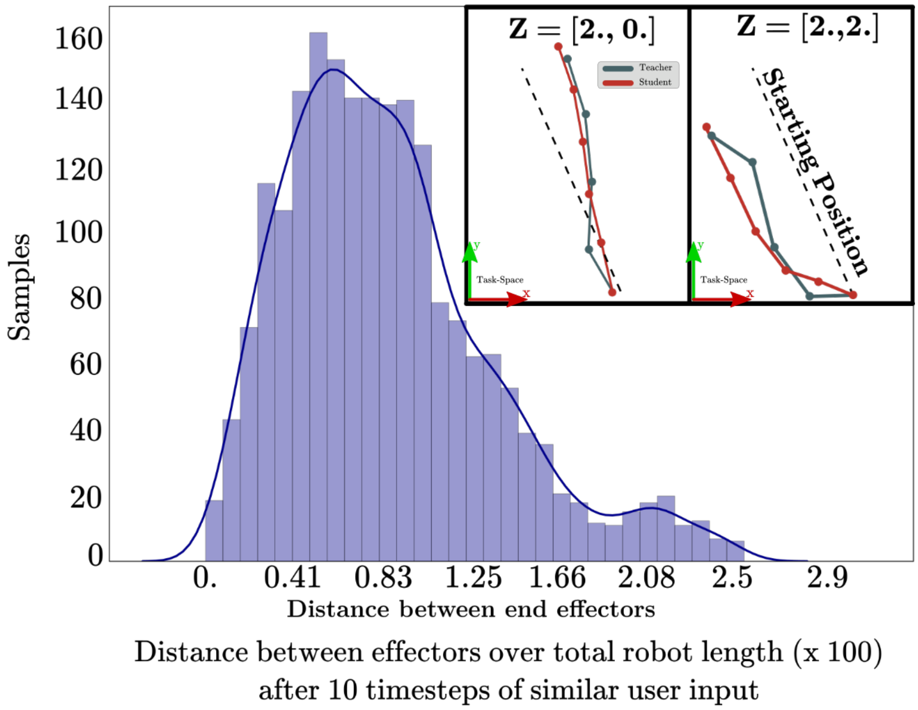


FIG. 5.22. Distribution of the distance between effectors for 1000 samples after 10 successive actions using the same user input z.

- The state $s \in \mathbb{R}^9 = [s_{\text{robot}}, s_{\text{ball}}, P_{\text{wall}}]$ where:
  - $s_{\text{robot}} \in \mathbb{R}^4$ holds information about the robot's joints angles
  - $s_{\text{ball}} \in \mathbb{R}^4$ describes the relative ball position and its current speed
  - $P_{\text{wall}} \in \mathbb{R}$ is the vertical target position on the wall used to enhance the distribution of impacts and ultimately have more control over the expert trajectories dataset
- The action $a \in \mathbb{R}^4$: angular speed for each joint
- The reward $r_{\text{tennis}} = \alpha + c \times (\beta + \gamma * \exp(-d))$ where:
  - α is a small constant that incites the agent to keep the ball over a height threshold for as long as possible
  - c is the contact flag. Its value ranges from 0 to 1 when a contact between the ball and the wall is detected, and goes back to 0 at the next step
  - β and γ are constant values used to weight the relative importance of accuracy when touching the wall.
  - d is the vertical offset between the ball and the target on the wall

After 10^5 episodes of training, amounting to 4 hours, the 4R agent is effectively able to play and can center its shots around the expected target. Figure 5.23 shows the evolution of the mean reward through training on the right, and the impact distribution of 100 shots per target for 6 different targets on the left. As we can see, the agent manages to direct the ball around the target and most of the shots land in the vicinity of the expected impact point.

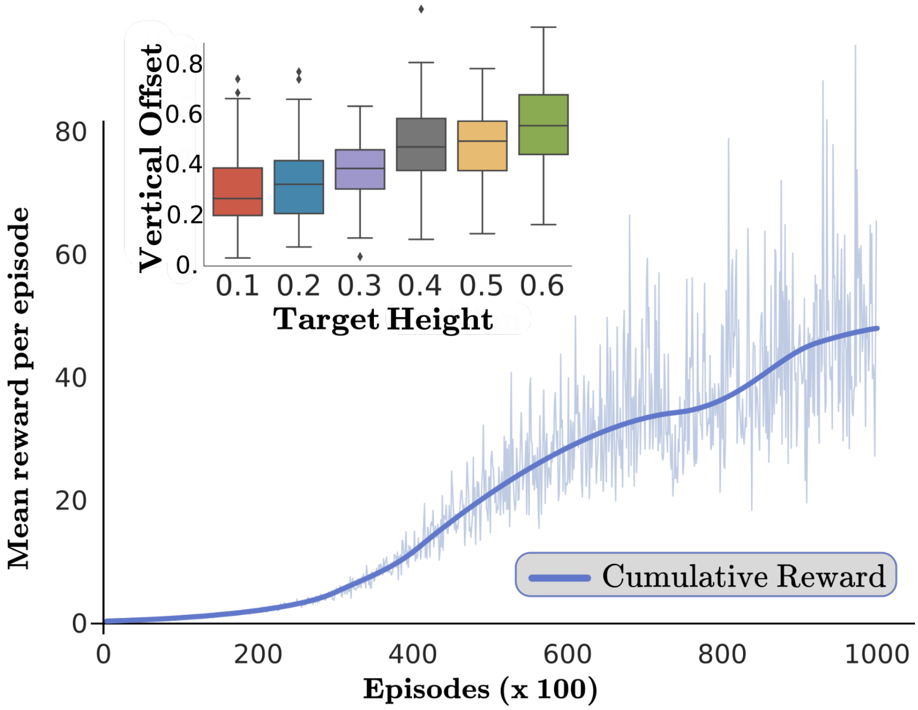


FIG. 5.23. Mean Cumulative Reward Evolution and ball impact distribution for 100 shots per target point.

Once the expert is available, the process is similar to the **Circle** task. To train the teacher agent, a balanced (with respect to the impact points) expert trajectories dataset is generated. The teacher VAE model optimizes its parameters to be able to recover the action, while encoding the inputs in a normally-distributed latent representation. In this specific case, the KL loss weight is halved to ensure a better reconstruction. Figure 5.24 shows the evolution of the reconstruction loss for the training along the epochs, as well as the distribution of the latent encodings for a batch of 2048 examples. Eventually, using the TSL defined by Equation 5.12, the task knowledge is distilled within two students: a 5R and a 3R robot.

In this configuration, we assess the controllability and efficiency of the trained student by relying on 4 human players. In our setting, each human player plays five games with each robot in the **Tennis** environment. For each game, we record the number of bounces against the wall. The objective is to maximize the number of bounces. The results of this evaluation are displayed in Figure 5.25. As a baseline, we add a plot for the expert, as well as a randomly initialized student agent, for both the 5R and the 3R, controlled by the best human player P0. Although a certain variability can be observed between the players, relative to their personal ability, each of them is able to reach much higher scores

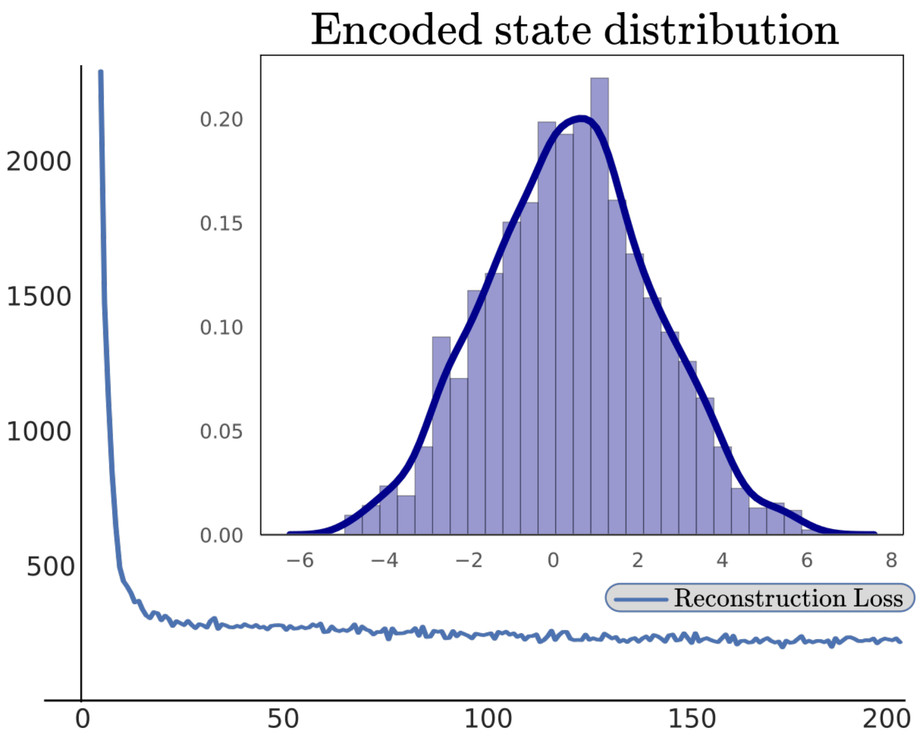


FIG. 5.24. Tennis Teacher training metrics and code distribution of a randomly sampled batch of 2048 examples.

than the random baseline, with both students and the teacher. After switching assistive agents, from teacher to students, most players are able to conserve their average score, except P0 which average score decreases of approximately 15%. However, even these lesser scores are still considerably beyond the random baseline.

This illustrates clearly that transfer through the TSL is effective. Additionally, the four players even decrease the variance of their score when using the student agent, underlining the robustness of the process. However, as it is difficult to assess these performances from sheer numerical values, a link to a video of our results is provided at the end of this chapter.

5.2.5.3 Autonomous TSL

Although the TSL assistive aspect is an interesting feature, particularly suited to numerous configurations in which it is desirable to rely on an external operator to provide the control signal and supervise the ongoing operation, there exist setups in which the agent autonomy is an even more desirable facet. This raises interrogations about the meaning of this autonomy. Indeed, in the context of a task such as the **circle-to-circle displacement**, the agent obviously requires an external incentive to move to a given point, thus reducing the interest of an unsupervised agent in the TSL framework. However, the **Tennis** environment yields a more exciting perspective, TSL-wise. In fact, in this task, an autonomous agent could maintain the system in a controlled state, specifically bouncing the ball along an unchanged trajectory, when a user-controlled robot could produce more

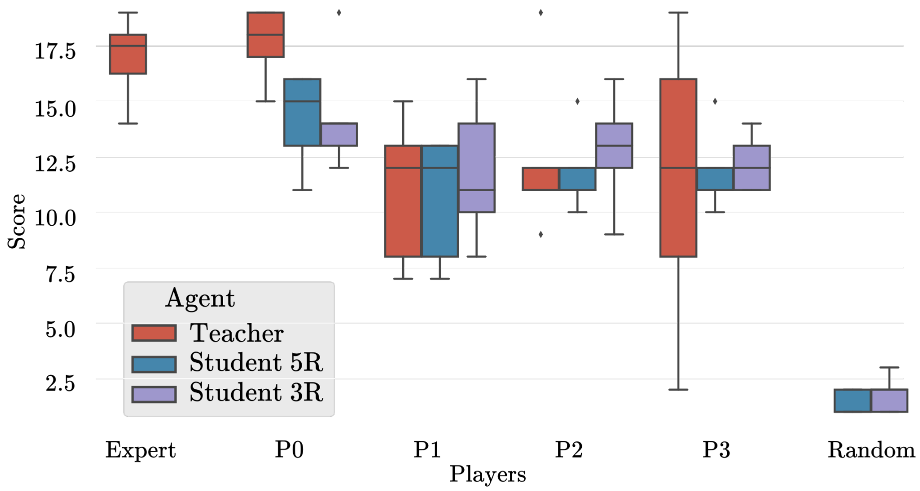


FIG. 5.25. Comparison of assistive performances of several players using both teacher and student agent with baseline.

diverse movements, as schematized in Figure 5.26.

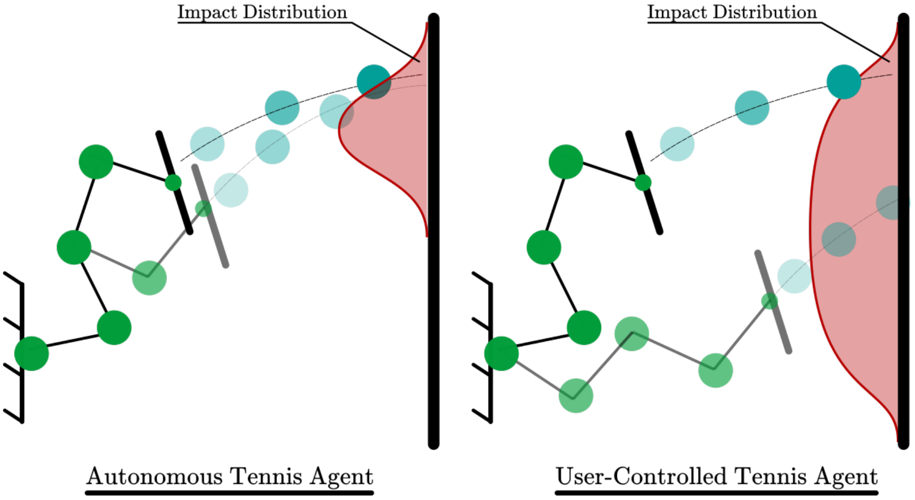


FIG. 5.26. Comparison of TSL-based controlled for autonomous agent and user-controlled agent in a relevant environment.

As was shown in the previous experiments, and particularly in the **circle-to-circle displacement**, the user input magnitude influences the output action magnitude. As such, due to the current specificities of the TSL formulation, endowing an agent with the autonomous characteristics, in a stabilization sense as defined above, can be done by ensuring the user signal magnitude is null. In practice, this means that a vector of zeros z_0 is concatenated with the state s and fed in the student decoder. That is:

$$a_{\text{Auto}} = D(z_0, s) \tag{5.13}$$

where $z_0 \in \mathbb{R}^{\dim(z)}$.

To evaluate the student agent ability to stabilize the ball without user direct supervision and control, the impact position distribution and the action magnitude are recorded along several episodes and are displayed in Figure 5.27. As can be observed, every run shown in this figure exhibits globally the same partition: at the beginning of the episode, the agent outputs aggressive actions to catch the ball and orient its velocity in a way that it finds fitting. Once in this configuration, the agent succeeds in maintaining the ball for a while with a sequence of actions of lower magnitude. Finally, a third phase can be detected when the agent misses the ball and resumes to a more aggressive control to catch it back. Meanwhile, the histogram representing the distribution of impacts for each episode shows that the agent is able to hit the ball so that most rebounds are located in a restricted area, thus underlining the stabilizing feature of the autonomous TSL.

5.2.5.4 Training Time & Assistive aspects

One of the principal motivation for transfer learning is to decrease the time needed to produce a functional model. The TSL approach clearly fits within this framework, as the effective transfer process is quite short on a regular i7 CPU with 64 Gb of RAM. In the **Circle** case, as the expert dataset relies on an inverse kinematics approach, the time needed for generating expert trajectories is negligible. As a result, a user could consider the fact to create an expert with the student structure as well. However, as the teacher training time using the expert data is close to 15 minutes, the TSL approach nevertheless yields a 66% time reduction in this environment as the transfer only requires 5 minutes. While it may appear of limited interest given the overall duration of the process, this simple environment allowed us to illustrate the TSL capabilities. The transfer significance is more strongly emphasized within the **Tennis** environment. In this configuration, the RL agent interacts during 4 hours with the environment before yielding a suitable expert policy, to which additional 15 minutes are necessary to train the teacher agent. In this case, the transfer using the TSL approach can be done in 5 minutes, that is only 7% of the time initially needed for the teacher, and yet results in functionning assistive controllers as demonstrated by our results. This clearly establishes the transfer value and shows that the more the expert dataset generation is costly, the more interesting the transfer of control policies is.

From a qualitative standpoint, this approach yields an important ease-of-use enhancement, particularly in the case of the Tennis task. Indeed, it would be highly challenging and non-intuitive to design and use a controller to manually pilot each rotoid joint of the tennis agent. The assistive method, empowered by the TSL method, allows users to easily reach a considerable number of bounces by letting the user focus on a global plan, instead of managing the low-level inputs.

5.2.6 Conclusion on Task-Specific Loss

In this section, it is proposed a new step by step methodology allowing to systematically transfer knowledge of task between two different structural agents. The introduced concept of Task-Specific Loss (TSL) was evaluated over various tasks, within the background of assistive robotics. As can be seen in the results, this method offers considerable time saving compared to state-of-the-art RL methods and can distill the task knowledge of a teacher agent within a structurally different student agent within minutes, as opposed to several hours of training for more complex task. Furthermore, our experiments, especially the Tennis task, demonstrate that the transferred knowledge is robust and that various users are able to generate stable performances on a task involving physics, planning and consistency. Additionally, this TSL formulation enables in some cases an autonomous usage, in a stabilization sense. Although improvable, these results were obtained without dedicated architecture or agent modification or fine-tuning. The TSL application scenarios are numerous: in assistive robotics, as it can be considered to allow disabled people to seamlessly fit their new robotic platform to behave similarly to their previous one. In other robotic domains as well, since learning-based transfer techniques for control tasks are not particularly developed.

5.3 Conclusion on TEacher-Centered methods

This chapter has presented two new approaches for transfer learning. These techniques can be considered as TEacher-Centered because, as opposed to the UNN method, they do not aim at embedding the task knowledge in a separate module but instead directly distill the expert behaviour within the new agent control module. Methods belonging to this family are appealing because they do not fix a particular interface between the agents and are thus more flexible. This flexibility enables more sophisticated losses and less restrictions over the operating agents. Furthermore, unsupervised training is usually more reliable than RL, making the transfer more robust. The main limitation of these approaches in their current form is a likely accuracy loss. Indeed, the very definition of VAE implies smooth Gaussian-like latent space, which makes it hard to have very precise values, a well known issue in literature. Similarly, the discriminator in the CoachGAN setting learns a probability distribution and a too sharp frontier makes it difficult for the generator to learn. However, these issues nevertheless allowed the transfer to occur for the presented tasks and can be mitigated by putting more care in the loss definition or dataset

Publications

- **CoachGAN: Fast Adversarial Transfer Learning between differently shaped entities** - International Conference on Informatics in Control, Automation and Robotics (ICINCO) 2020 - Paris, France

- **Task-Specific Loss: A TEacher-Centered Approach to Transfer Learning** - Submitted to: Lecture Notes in Electrical Engineering - Springer

Additional Material

- https://drive.google.com/drive/folders/1EZ0vSB5vzCOAbtHOPLlpOl8Megd0gm14?usp=sharing

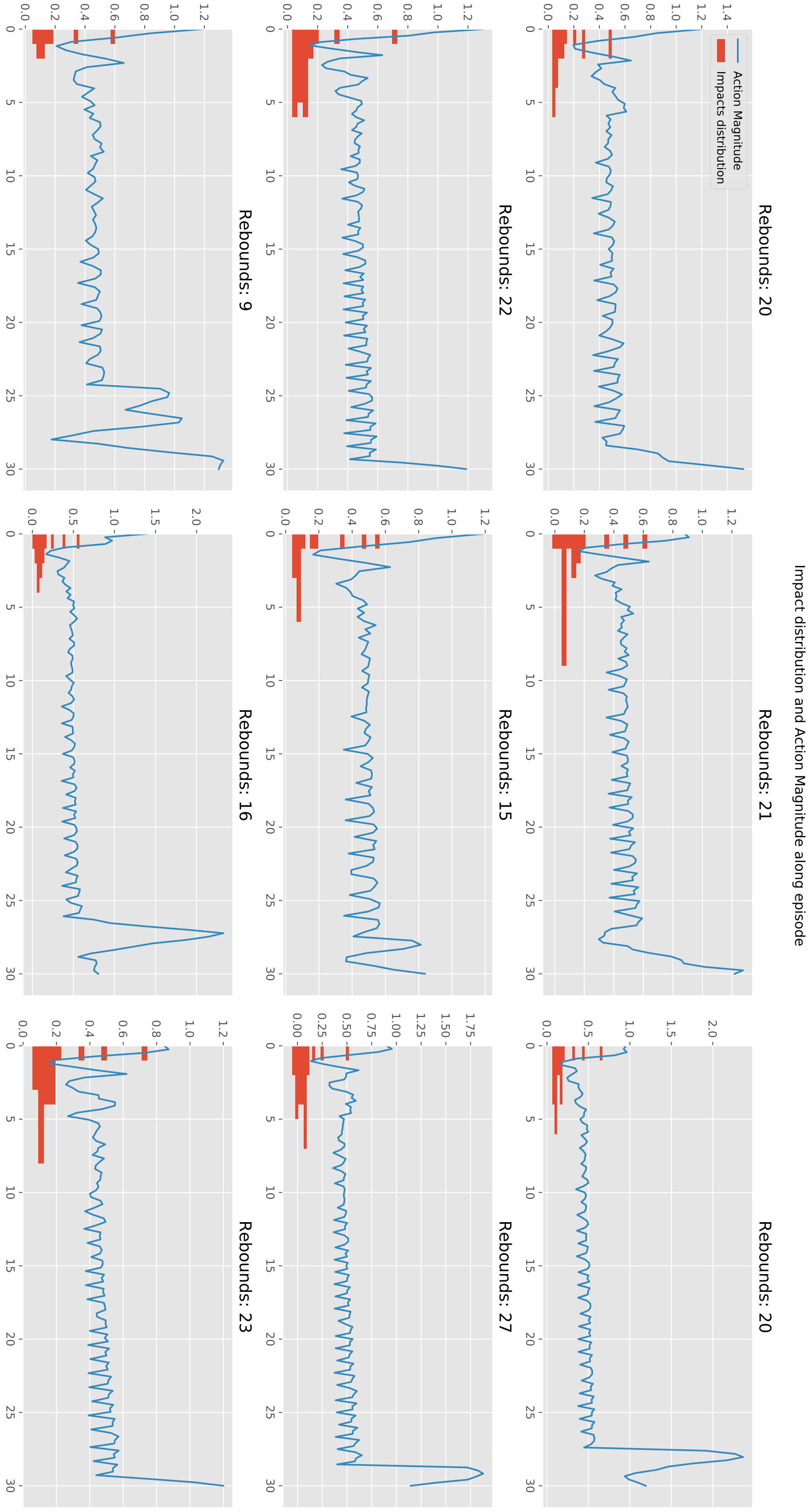


FIG. 5.27. Impact distribution and action magnitude of an autonomous student along several episodes.

Chapter 6

General Conclusion & Prospects

General Conclusion

As it appears reasonable to expect robots to endorse an ever more important role in the future human civilization, it is necessary to envision the constraints and modalities for managing a fleet of robots. Currently, several non-negligible and systemic issues stand in the way of the large-scale deployment of robots. In particular, the known sample inefficiency of learning-based approaches, in particular for on-policy Reinforcement Learning frameworks, and the lack of structured transfer-learning strategies for these agents are representative of these issues. Consequently, at the beginning of this thesis, we asked whether the transfer of task-specific knowledge between two entities was possible, despite their potential morphological differences. We considered how transfer is applied in human societies and were able to propose a classification of techniques: TAsk-Centered and TEacher-Centered transfer.

TAsk-Centered transfer techniques

TAsk-Centered techniques (Section 4.1) are inspired from a current common situation where a specialized piece of knowledge is embedded in a support without direct connections between the entity who generated the knowledge and the one receiving it. A clear human analogy could be learning skills from a video online. This method requires nevertheless a shared interface to successfully link the piece of knowledge with the entities: for instance, it is unrealistic to consider learning to juggle with four balls if only two are available.

From a practical point of view, this method was implemented in a Reinforcement RL pipeline. Having highlighted the fact that *vanilla* architectures and training procedures lack a clear knowledge structure, as opposed to bigger models with defined dimensionality bottlenecks, we proposed a staged training method that decouples the control policy of a given agent from the dedicated task skills. The numerous experiments setup to evaluate this idea provided various insights: first, the control policy pre-training yields, on its own, several improvements over naive initialization as it is possible to embed relevant movement strategies in the agent in a way that enables it to better tackles downstream tasks. As explained above, RL tends to produce agents with shaky, unstable behaviour and the introduction of additional reward terms designed to ensure smoother actions are liable to destabilize the policy altogether, consequently making these empirical results valuable.

Then, as demonstrated in the transfer between various agents (Section 4.2), the knowledge support module, called the **Universal Notice Network (UNN)**, does supply the transfer target with appropriate information on the task. Agents with a different kinematic structure and no previous training on the specific task are able to reach performances close to those of the agent that generated the knowledge, in only a fraction of the initial number of iterations to fine-tune their behaviours.

As mentioned, the UNN method depends on a shared interface defined at the beginning of the initial training process. This interface consequently creates ties between all new agents receiving the module and the initial agent, which will prevent agents kinematically unable to attach to this system to use the task knowledge.

TEacher-Centered Transfer Techniques

The **TEacher-Centered** family of techniques is the result of efforts directed towards the abstraction of the UNN interface. Indeed, instead of providing a restrictive state vector to which all subsequent agents must conform, TEacher-Centered approaches are based on the idea that, in order to increase its usability, the task logic should rely on a kinematically-independent metric to evaluate actions. Conceptually, in a glass-of-water-fetching task, while the UNN would provide the position and orientation of both effectors holding the glass (thus making the task logic inaccessible to robots with a single effector), TEacher-Centered methods would provide a target glass state and let the agent manage its kinematic configuration to reach such a state. To reach this objective, both TEacher-Centered procedures rely on a fundamental feature: the Intermediate State Variable. This vector, obtained through a differential computation, links the teacher agent with the student and enables the latter to optimize its behaviour to match the teacher's. The Intermediate State Variable can be seen as a more flexible and customizable Universal Notice Network interface.

The **Intermediate State Variable** (ISV) is central in the two TEacher-Centered techniques proposed. The first one, **CoachGAN** (Section 5.1.2), uses the Generative Adversarial Framework to create a discriminative model able to detect whether a given state is likely under the expert policy it has been trained on. Using the ISV permits to translate the student states, potentially with a kinematic structure different from the expert, into expert states, consequently enabling the discriminative teacher to provide an evaluation of the movement. As the ISV computation is differentiable, it is suitable for optimizing the student agent. In the experiments setup to evaluate the CoachGAN technique, multiple ISVs are used and we demonstrate that both analytical functions as well as approximations with neural networks are usable, thus largely widening the applicability of this method (Section 5.1.6).

The second approach, called the **Task-Specific Loss** (Section 5.2), addresses some of the CoachGAN limitations, such as the fact the latter procedure was constrained to states prediction. This alternative formulation, based on a Variational AutoEncoder architecture, provides a framework that compares the result of student and teacher actions in an ISV defined metric and penalizes the student agent when the result of its action results in a larger distance between its new ISV state and the teacher ISV. This procedure has been evaluated on several environments and, despite the simplicity of these environments, demonstrates that task knowledge can indeed be transferred through the differentiable ISV function. In addition, this approach enables the assistive control of an agent (Section 5.2.5).

These developments, in spite of the encouraging results obtained, are not complete and present undeniable limitations. First, most environments in the TEacher-Centered methods are not particularly complex and include assumptions that may fail if deployed in a less permissive system. While theoretically compatible with more sophisticated worlds, TEacher-Centered methods are yet to be evaluated in practical cases. Then, in both families of approaches, very few physical constraints are taken into account for the robots. This aspect is slightly less problematic in the case of TEacher-Centered methods, CoachGAN in particular, as it *only* requires looking for solutions in another area of the state space. However, this can constitute a serious obstacle in the UNN configuration as some states may not be reachable for another robot. Finally, an issue concerning more closely TEacher-Centered methods is the absence of clear re-training strategy. Indeed, while the TAsk-Centered UNN was based on RL, which can in principle overcome the initial teacher by enabling backpropagation to affect all the modules, TEacher-Centered procedures are tied to the expert performance and even in the perspective of a hardly attainable perfect transfer, current formulation simply does not conceive a way for the performance to exceed and go beyond the teacher.

Prospect

TEacher-Centered and TAsk-Centered methods convergence

The decoupling between task and policy, implemented in TAsk-Centered methods, presents various advantages, in particular in the transfer from simulation to reality. As a matter of fact, training a policy in a simulated environment and releasing it in the physical world is not likely to result in successes, partly because of state distribution disparities. Consequently, it is conceivable to protect the task logic, learned in simulation, from those unexpected states by using approximators that could translate these outlying states in the state space area in which the policy was initially trained. Furthermore, the output module could be trained solely in the real world to take into account the real robot properties. Doing so, in a primitive task as explained in this manuscript, is both safer and faster than training the entire agent on a more complex task from scratch.

Furthermore, a slight paradigm change could nonetheless open new multiple and exciting perspectives for future works based on the two main ideas developed in this manuscript. For instance, the BAM approach that was designed to prevent several issues related with interface compatibities between agents, presents, due to the model simplicity, an almost trivial training procedure. This technique can consequently easily generate experts in a given environment but needs to ensure continuity between the downstream sophisticated agents and the relatively constraints-free BAM agent, which is the TAsk-Centered method weakness. This setup present multiple opportunities for implementing TEacher-Centered methods where instead of managing complex interfaces and normalization parameters between morphological distinct agents, an ISV metric could be used to distill the BAM expert knowledge within the student. Given the BAM expert simplicity, it would even be conceivable to include analytical approaches when relevant, thus vastly extending the scope of this method.

Merging TEacher-Centered and TAsk-Centered methods with external works

There are advantages in merging these transfer strategies with other recent works. Calling in external methods, it could be, for instance, convenient to define a UNN formulation in the scope of Graph Neural Networks. This way, each agent joint/part would be considered as a node graph, training could then be targeted towards a single joint and then passed to every other segment, thus extending the method flexibility.

Other procedures using graphs could be foreseen as well. For instance, the UNN could be formulated as a GAN framework and trained to generate images based on the current agent and world state. These images could be considered as mazes that would then be solved by a search algorithm, such as $A^\star$. A loss function would take into account whether the agent was able to reach its goal and penalize the UNN for inaccurate or unpractical graphs, in an adversarial fashion. Doing so, again using ISV, would provide reusability of the UNN to different agents.

By stepping back and having a look at the current research landscape, recent works in Off-Policy Reinforcement Learning have highlighted goals and practical difficulties that share multiple features with the works described in this PhD manuscript. More precisely, considerable efforts are invested in this field toward the creation of a framework in which an agent can efficiently reuse experiences and trajectories generated by various policies to predict an accurate value function and ultimately train a successful policy. As opposed to on-policy approaches that systematically require fresh samples, these techniques present the advantage of theoretically reducing the number of interactions needed between the agent and the environment.

As explained, this setup conceptually mirrors the ones developed in the contributions of this thesis, as it expects to reuse the knowledge of other agents, experts as well as sub-optimal policies (embodied in the same agent in the case of off-policy RL) and devise through the trajectories present in the dataset a value function or an action value function that is then used to guide and fit the policy.

In this view, instead of attempting to supplant these efforts to enable off-policy agents to learn from other agents experiences, an interesting way would be to position the Teacher-Centered insights and techniques at a future convergence point. Specifically, TEC approaches would allow a given agent to be trained on data generated by another policy embedded in an agent with a different structure. The implementation of this process relies mainly on ISV and can be envisioned in one of two ways:

- Transform states from the expert trajectories into the new agent states using ISV. Learn the new agent state value function from these ISV-translated states and fit the policy as in usual off-policy frameworks.

- Learn a relevant value function on the expert trajectories first. For the new agent, for a given state, each action taken will result in a new state that can be translated using ISV, and evaluated by the original expert value function.

While ultimately pursuing the same goal, the selection between these two options will likely take into account the reliability and sensibility of the methods used to compute the value function and the ISV. For instance, an overfitted value function, displaying extreme sensibility to the initial dataset may fail with given states coming from a loose ISV. In this case, translating the expert dataset using ISV might be more relevant. As explained above, despite the existence of several technical points to answer to and design choices to infer, we are convinced that this modification to the Teacher Centered framework could provide an additional flexibility to the method and ultimately enhace its applicability.

Bibliography

[1] Bacon, P., Harb, J., and Precup, D. The option-critic architecture. *Association for Advancement of Artificial Intelligence (AAAI)* (Feb 2016).

[2] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent Tool Use From Multi-Agent Autocurricula. *International Conference on Learning Representations (ICLR)* (Nov 2020).

[3] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. Emergent complexity via multi-agent competition. *International Conference on Learning Representations (ICLR)* (2018).

[4] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems* (2016).

[5] Bellman, R. E. *Dynamic Programming.* Dover Publications, Incorporated, 2003.

[6] Berkeley, G. *Three Dialogues Between Hylas and Philonous.* G. James, 1713.

[7] Berkenkamp, F., and Schoell, Angela, P. Safe and robust learning control with gaussian processes. *European Control Conference (EEC)* (July 2015).

[8] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

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

[10] Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. Exploration by random network distillation. *International Conference On Representation Learning (ICRL)* (2018).

[11] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. Understanding disentangling in β-VAE. *International Conference on Representation Learning (ICLR)* (Apr. 2018).

[12] Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G. S., Schaal, S., and Levine, S. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *International Conference on Machine Learning (ICML)* (2017).

[13] Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)* (2020).

[14] Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *International Control on Machine Learning (ICML)* (2018).

[15] Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. *Proceedings of Machine Learning Research (MLR)* (2018).

[16] Conneau, A., and Lample, G. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada* (2019), pp. 7057–7067.

[17] Cooper, S. B., and Leeuwen, J. v. *Alan Turing: His Work and Impact*, 1st ed. Elsevier Science, San Diego, CA, USA, 2013.

[18] Copeland, B. J. *Artificial intelligence.* Encyclopædia Britannica, Mar 2020.

[19] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS) 2*, 4 (Dec. 1989), 303–314.

[20] Deisenroth, M. P., and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (Madison, WI, USA, 2011), ICML'11, Omnipress, p. 465–472.

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

[22] Diehl, M., Bock, H. G., Diedam, H., and Wieber, P.-B. Fast direct multiple shooting algorithms for optimal robot control. In *Fast Motions in Biomechanics and Robotics*, M. Diehl and K. Mombaur, Eds., vol. 340 of *Lecture Notes in Control and Information Sciences.* Springer Berlin / Heidelberg, 2006, pp. 65–93.

[23] Domingos, P. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World.* Basic Books, Inc., USA, 2018.

[24] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. *Proceedings of Machine Learning Research (MLR)* (2014).

[25] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL2: Fast reinforcement learning via slow reinforcement learning. *International Conference on Representation Learning (ICRL)* (2017).

[26] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *CoRR abs/1901.10995* (2019).

[27] Edwards, A. D., Sahni, H., Schroecker, Y., and Jr., C. L. I. Imitating latent policies from observation. *Proceedings of Machine Learning Research (MLR)* (2019).

[28] Elman, J. L. Learning and development in neural networks: The importance of starting small. *Cognition 48* (1993).

[29] Englert, P., and Toussaint, M. Combined optimization and reinforcement learning for manipulation skills. *Robotics: Science and Systems* (2016), 137–154.

[30] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012).

[31] Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *International Conference on Representation Learning (ICRL)* (2019).

[32] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning (ICML)* (2017).

[33] Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. Meta-learning shared hierarchies. *International Conference On Representative Learning (ICRL)* (October 2017).

[34] Gatys, L. A., Ecker, A. S., and Bethge, M. Image Style Transfer Using Convolutional Neural Networks. *Computer Vision and Pattern Recognition (CVPR)* (Aug. 2016).

[35] Ghazanfari, B., Afghah, F., and Taylor, M. E. Sequential association rule mining for autonomously extracting hierarchical task structures in reinforcement learning. *IEEE Access 8* (2020), 11782–11799.

[36] Girshick, R. Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)* (USA, 2015), IEEE Computer Society, p. 1440–1448.

[37] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *Computer Vision and Pattern Recognition (CVPR)* (2014).

[38] Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J., and Waytowich, N. R. Integrating behavior cloning and reinforcement learning for improved performance in sparse reward environments. *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (2020).

[39] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[40] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *International Conference on Neural Information Processing Systems (NeurIPS)* (2014).

[41] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. Continuous deep-q learning with model-based acceleration. *International Conference on Machine Learning (ICML)* (March 2016).

[42] Ha, D., and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 2450–2462.

[43] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. *International Conference on Machine Learning (ICML)* (2017).

[44] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *International Conference on Machine Learning (ICML)* (2018).

[45] Hassoun, M. H. *Fundamentals of Artificial Neural Networks*, 1st ed. MIT Press, Cambridge, MA, USA, 1995.

[46] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations* (2018).

[47] Haykin, S. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.

[48] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition (CVPR)* (2015).

[49] Heess, N., Wayne, G., Tassa, Y., Lillicrap, T. P., Riedmiller, M. A., and Silver, D. Learning and transfer of modulated locomotor controllers. *CoRR abs/1610.05182* (2016).

[50] Ho, J., and Ermon, S. Generative adversarial imitation learning. *International Conference on Neural Information Processing Systems (NeurIPS)* (2016).

[51] Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *CoRR abs/1605.09674* (2016).

[52] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR abs/1704.04861* (2017).

[53] Howard, J., and Ruder, S. Universal Language Model Fine-tuning for Text Classification. *Association for Computational Linguistics (ACL)* (2018).

[54] Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. *Computer Vision and Pattern Recognition (CVPR)* (2017).

[55] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *International Conference on Representation Learning (ICRL)* (2017).

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

[57] Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. Procedural level generation improves generality of deep reinforcement learning. *Association for Advancement of Artificial Intelligence (AAAI)* (2018).

[58] Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uskoreit, J. One model to learn them all. *arXiv:170605137v1* (June 2017).

[59] Kalawoun, R., Lengagne, S., Bouchon, F., and Mezouar, Y. Bsplines properties with interval analysis for constraint satisfaction problem: Application in robotics. In *15th International Conference on Intelligent Autonomous Systems IAS-15* (june 2018).

[60] Kalawoun, R., Lengagne, S., and Mezouar, Y. Optimal robot base placements for coverage tasks. In *14th IEEE International Conference on Automation Science and Engineering (CASE 2018)* (August 20-24 2018).

[61] Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review 53* (2020).

[62] Kingma, D. P., and Welling, M. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR Banff, AB, Canada, April 14-16, Conference Track Proceedings* (2014).

[63] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (Red Hook, NY, USA, 2012), NIPS'12, Curran Associates Inc., p. 1097–1105.

[64] Kullback, S., and Leibler, R. A. On information and sufficiency. *Ann. Math. Statist. 22*, 1 (03 1951), 79–86.

[65] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations* (2020).

[66] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE* (1998), pp. 2278–2324.

[67] Lengagne, S., Vaillant, J., Kheddar, A., and Yoshida, E. Generation of whole-body optimal dynamic multi-contact motions. *International Journal of Robotics Research* (Apr 2013), 17.

[68] Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. State Representation Learning for Control: An Overview. *Neural Networks 108* (Dec. 2018), 379–392.

[69] Lesort, T., Seurin, M., Li, X., Rodríguez, N. D., and Filliat, D. Unsupervised state representation learning with robotic priors: a robustness benchmark. *CoRR abs/1709.05185* (2017).

[70] Levine, S. Lecture notes in CS 294-112: Deep reinforcement learning, 2016.

[71] Lilicrap, T., Hunt, J., Pritzel, A., Hess, N., Erez, T., Silver, D., Tassa, Y., and Wiestra, D. Continuous control with deep reinforcement learning. *International Conference on Representation Learning (ICRL)* (February 2016).

[72] Little, D. Y., and Sommer, F. T. Learning in embodied action-perception loops through exploration. *CoRR abs/1112.1125* (2011).

[73] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized BERT pretraining approach. *CoRR abs/1907.11692* (2019).

[74] Lopes, M., Lang, T., Toussaint, M., and yves Oudeyer, P. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 206–214.

[75] Losey, D. P., Srinivasan, K., Mandlekar, A., Garg, A., and Sadigh, D. Controlling Assistive Robots with Learned Latent Actions. *arXiv e-prints* (Sep 2019), arXiv:1909.09674.

[76] Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. Learning latent plans from play. *CoRR abs/1903.01973* (2019).

[77] Martin, L., Muller, B., Ortiz Suárez, P. J., Dupont, Y., Romary, L., Villemonte de La Clergerie, É., Seddah, D., and Sagot, B. CamemBERT: a Tasty French Language Model. Web site: https://camembert-model.fr, Oct. 2019.

[78] Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. Unrolled generative adversarial networks. *CoRR abs/1611.02163* (2016).

[79] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Red Hook, NY, USA, 2013), NIPS'13, Curran Associates Inc., p. 3111–3119.

[80] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119.

[81] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing Atari with Deep Reinforcement Learning. *CoRR abs/1312.5602* (2013).

[82] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (Feb. 2015), 529–533.

[83] Mohamed, S., and Jimenez Rezende, D. Variational information maximisation for intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems* (09 2015), 2772–2782.

[84] Mounsif, M., Lengagne, S., Thuilot, B., and Adouane, L. Universal Notice Network: Transferable Knowledge Among Agents. *6th 2019 International Conference on Control, Decision and Information Technologies (IEEE-CoDIT)* (Apr. 2019).

[85] Mounsif, M., Lengagne, S., Thuilot, B., and Adouane, L. BAM! Base Abstracted Modeling with Universal Notice Network:Fast Skill Transfer Between Mobile Manipulators. *7th 2020 International Conference on Control, Decision and Information Technologies (IEEE-CoDIT)* (July 2020).

[86] Mounsif, M., Lengagne, S., Thuilot, B., and Adouane, L. CoachGAN: Fast Adversarial Transfer Learning between Differently Shaped Entities. *17th International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (2020).

[87] Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. *International Conference on Neural Information Processing Systems (NeurIPS)* (2017).

[88] Nagabandi, A., Kahn, G., Fearing, Ronald, S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *University of California Berkeley Report* (August 2017).

[89] Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. *International Conference on Representation Learning (ICRL)* (2019).

[90] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. P. Action-conditional video prediction using deep networks in atari games. *International Conference on Neural Information Processing Systems (NeurIPS)* (2015).

[91] OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving Rubik's Cube with a Robot Hand. *arXiv e-prints* (Oct 2019), arXiv:1910.07113.

[92] Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. *International Conference on Neural Information Processing Systems (NeurIPS)* (2018).

[93] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[94] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. *International Conference on Machine Learning (ICML)* (2017).

[95] Paul, C., Zain, S., Igor, M., Jonas, S., Trevor, B., Joshua, T., Pieter, A., and Wojciech, Z. Transfer from simulation to real world through learning deep inverse dynamics model. *OpenAI - University of California - Berkeley* (October 2016).

[96] Peng, B., Li, X., Gao, J., Liu, J., and Wong, K.-F. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 2182–2192.

[97] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. *International Conference on Robotic and Automation (ICRA)* (2018).

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

[99] Peng, X. B., Chang, M., Zhang, G., Abbeel, P., and Levine, S. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 3681–3692.

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

[101] Peng, X. B., Kumar, A., Zhang, G., and Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *CoRR abs/1910.00177* (2019).

[102] Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. Asymmetric actor critic for image-based robot learning. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018* (2018), H. Kress-Gazit, S. S. Srinivasa, T. Howard, and N. Atanasov, Eds.

[103] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *International Conference on Learning Representation (ICLR)* (2018).

[104] Radford, A. Improving language understanding by generative pre-training. *OpenAI Report* (2018).

[105] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Report* (2019).

[106] Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. CNN features off-the-shelf: an astounding baseline for recognition. *Computer Vision and Pattern Recognition (CVPR)* (2014).

[107] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection, 2016.

[108] Redmon, J., and Farhadi, A. YOLO9000: better, faster, stronger. *Computer Vision and Pattern Recognition (CVPR)* (2017).

[109] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *CoRR abs/1804.02767* (2018).

[110] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99.

[111] Riedmiller, M. A., Hafner, R., Lampe, T., Neunert, M., Degrave, J., de Wiele, T. V., Mnih, V., Heess, N., and Springenberg, J. T. Learning by playing - solving sparse reward tasks from scratch. *International Conference on Learning Representation (ICLR)International Conference on Machine Learning (ICML)* (2018).

[112] Ronneberger, O., P.Fischer, and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (2015), vol. 9351 of *LNCS*, Springer, pp. 234–241. (available on arXiv:1505.04597 [cs.CV]).

[113] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, USA, 2009.

[114] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2016), NIPS'16, Curran Associates Inc., p. 2234–2242.

[115] Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *Computer Vision and Pattern Recognition (CVPR)* (2018).

[116] Schmidhuber, J. *Evolutionary Principles in Self-Referential Learning.* PhD thesis, Technische Universitat München, 1987.

[117] Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats* (1991).

[118] Schmidhuber, J. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development 2*, 3 (2010), 230–247.

[119] Schulman, J., Abbeel, P., and Chen, X. Equivalence between policy gradients and soft q-learning. *CoRR abs/1704.06440* (2017).

[120] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. Trust region policy optimization. *International Conference on Machine Learning (ICML)* (April 2015).

[121] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representation* (September 2016).

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

[123] Schwab, K. *The Fourth Industrial Revolution.* Crown Publishing Group, 2017.

[124] Shelhamer, E., Long, J., and Darrell, T. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell. 39*, 4 (Apr. 2017), 640–651.

[125] Silver, D. Lecture notes in reinforcement learning: Policy gradient methods, December 2015.

[126] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittiwiser, J., Antonoglu, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sustkever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *The Journal of Nature* (October 2015).

[127] Silver, D., Simonyan, K., Antonoglu, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Adrian, B., Chen, Y., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. *The Journal of Nature* (April 2017).

[128] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations* (2015).

[129] Smith, L., Dhawan, N., Zhang, M., Abbeel, P., and Levine, S. AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos. *International Conference on Robotics Science and Systems (RSS)* (2020).

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

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

[132] Sutton, R. S. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull. 2*, 4 (July 1991), 160–163.

[133] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (2017), AAAI'17, AAAI Press, p. 4278–4284.

[134] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *Computer Vision and Pattern Recognition (CVPR)* (2016).

[135] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. *Computer Vision and Pattern Recognition (CVPR)* (2016).

[136] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1–9.

[137] Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR abs/1903.12136* (2019).

[138] Turing, A. M. Computing machinery and intelligence. *Mind 59*, 236 (1950), 433–460.

[139] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, u., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 6000–6010.

[140] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *International Conference on Machine Learning (ICML)* (2017).

[141] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *CoRR abs/1611.05763* (2016).

[142] Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *CoRR abs/1901.01753* (2019).

[143] Watson, I. The aims of artificial intelligence: A science fiction view. *IEEE Intelligent Systems 18*, 2 (Mar. 2003), 78–80.

[144] Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. A. Embed to control: A locally linear latent dynamics model for control from raw images. *International Conference on Neural Information Processing Systems (NeurIPS)* (2015).

[145] Xavier, G., and Bengio, Y. Understanding the difficulty of training deep feedforward neural network. *International Conference on Artificial Intelligence and Statistics (AISTATS)* (Mar. 2010).

[146] Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. *Computer Vision and Pattern Recognition (CVPR)* (2017).

[147] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 5753–5763.

[148] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? *International Conference on Neural Information Processing Systems (NeurIPS)* (2014).

[149] Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta-reinforcement learning, 2019.

[150] Zagoruyko, S., and Komodakis, N. Wide residual networks. *British Machine Vision Conference* (2016).

[151] Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *Computer Vision and Pattern Recognition (CVPR)* (2017).

[152] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *International Conference on Computer Vision (ICCV)* (2017).

[153] Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. *Association for Advancement of Artificial Intelligence (AAAI)* (2008).

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

Appendix A

UNN in Mobile Manipulators settings

Introduction

In order to further demonstrate the capabilities of the UNN/BAM method, environments featuring several mobile manipulators are considered. Using three mobile bases (omnidirectional, bicycle and differential), four robotic arm configurations and two environments (**Pick'n Place** task and the **Stacking** task), the RL experiments described below underline the advantages linked with the concept of task knowledge. These advantages, already introduced in the Task-Centered Chapter are twofold: firstly, it is beneficial to the initial learning process, as it eases the reward function design and prevents the RL agent to fall within local minima. Next, it is particularly suitable for transferring skills between differently shaped entities, outperforming *naive* transfer both in final performances and training time perspectives and even enabling zero-shot transfer in some particular cases.

Robots & Environments

Robots

We consider three mobile bases configurations (Omnidirectional, bicycle and differential), as well as four different manipulator types (KUKA, BLUE, Generic 2, Generic 3). Which, in total, leaves us with 12 possible configurations, three of them being displayed in Figure A.1. The bicycle mobile base is non-holonomic and relies on two values: thrust and steering, for moving. The differential base is also non-holonomic and uses two torque values to compute the resulting direction. Finally, the omnidirectional base is the simplest as it is holonomic and the two values determine linearly its next position. Concerning the manipulators, both the KUKA and BLUE robots have 5 DoF. The Generic arms (rightmost configuration in Figure A.1) have 2 DoF per articulation, which leads to a total of 4 and 6 DoF for the agents considered through our experiments. All serial manipulators are controlled using joints target position.

Environments

This section presents the two environments, implemented in the ML-Agents framework (introduced in (56)), that were created in order to demonstrate the UNN and the BAM advantages. An additional environment, the **Reaching** environment, is used to train the base modules and is

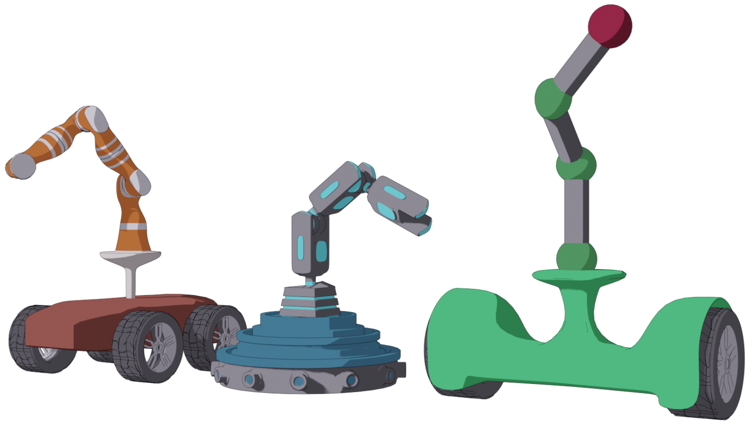


FIG. A.1. A subset of the possible agent configurations. From left to right: Bicycle KUKA, Omnidirectional BLUE, Differential G3. Parts can be exchanged between agents.

presented below. For each of these environments, we explain the general goal, define the task-related states and detail the extrinsic reward signal used to guide the policy in the environment. In these environments, the UNN expects a vector $s^{i'}$ from its input base and always outputs the same type of information o^{out}. In practice, we have:

- The processed intrinsic information $s^{i'} \in \mathbb{R}^6$ consist of the end effector linear position and velocity, computed by an analytical approach. Specifically, the input base relies on a forward kinematic approach to compute the processed intrinsic information.
- The UNN outputs $o^{out} \in \mathbb{R}^3 \times \mathbb{B}$, composed of the desired effector linear velocity and a Boolean for sucker action.

Reaching Task: This is the primitive base module training environment. Consequently, the state and actions of this specific task are agent-related. This task involves touching a given point in space with the agent's effector. The target point is regularly moved in order to force the agent to use its mobile base to get in range. This task features the following MDP:

- State: $s_t \in \mathbb{R}^{16+n}$: mobile base pose $\in \mathbb{R}^3$, arm pose $\in \mathbb{R}^n$ (n joints positions), vector to target $\in \mathbb{R}^3$, LIDAR sensors $\in \mathbb{R}^{10}$
- Action: $a_t \in \mathbb{R}^{2+n}$: mobile base movement vector $\in \mathbb{R}^2$ (specific to each mobile base configuration) and target joints speed $\in \mathbb{R}^n$.
- Reward: $r_t = -\alpha \times d_{E,T} + \sum_i \beta_i \times c_i$, where $d_{E,T}$ is the distance between the effector and the target, α is a normalizing constant and $\beta_i \times c_i$ is a weighted constraint on the arm to prevent it from falling into undesired local minima

For training the UNN and evaluating the BAM approach, we created two custom environments.
Pick'n Place Task: the agent is expected to pick up an object from one table and drop it to a given point in space. In this case, the task-related state is $s_{pap} \in \mathbb{R}^6 \times \mathbb{B}$, composed of the current object position, the drop target position and a boolean for signaling whether the cube is held or not. The agent relies on the following reward function:

$$r_p = \alpha(d_0 + d_1)^{-1} \tag{A.1}$$

where the reward r_p for the **Pick'n Place** environment depends on α a positive scaling factor, d_0 the distance from the object to the effector and d_1 the distance from the object to the drop target position.

Stacking Task: the goal is to stack a number of cubes. The task-related state is conceptually similar to the Pick'n Place, that is $s_{\text{stacking}} \in \mathbb{R}^{3*c} \times \mathbb{B}^4$ where c is the number of cubes to stack, in our case $c = 4$. We also design a reward that penalizes the agent according to the summed distances between each of the cube and their ideal position. That is:

$$r_s = [\sum_{i=1}^{4} \alpha_i (d_{0,i} + d_{1,i})]^{-1} \tag{A.2}$$

where the **Stacking** task reward r_s depends on α_i a positive scaling factor that is adjusted according to the agent progression in the task, $d_{0,i}$ the distance from the cube i to the effector and $d_{1,i}$ the distance from the cube i to its target position.

Training setup

While RL excels in reactive environments, it is usually less straightforward to design reward functions for sequential environments like those presented above. Indeed, in these cases, tuning the reward function in order to balance between a negative reward that may induce divergence in the agent's behaviour and a positive reward that may lead the agent to accumulate rewards without a clear line of action is very subtle and requires a considerable amount of tweaking. Thus, in order to improve the learning performances, PPO is coupled with two additional techniques:

- Pre-training with a Behavior Cloning based (38) approach
- Inverse Reinforcement Learning (50)

In practice, the agent weights go through a first training phase that consists of Behaviour-Cloning pre-training. Namely, this initial process is a supervised training setup where the agent error corresponds to the distance between its action prediction and the expert output. Then, during the Reinforcement Learning process, the agent loss function is enhanced with the Inverse Reinforcement Learning component. Specifically, this additional loss function evaluates how likely was the latest agent trajectory to be generated by expert policy. This additional loss is minimized when the agent's sequences of states and actions are similar to the expert's, thus providing a pedagogic signal to the training agent.

Results

In this section, the results of our method on both training and testing setup, over all environments are presented. In particular, we compare our architecture, the BAM modeling, with several baselines, namely the *naive* PPO approach and the UNN segmentation. A video of our results is available at `https://bit.ly/324Sxnz`.

Training

During training, we monitor the agent's evolution through the mean cumulative reward per episode. This metrics is common in RL and serves as a measure of how well the agent behaves within its

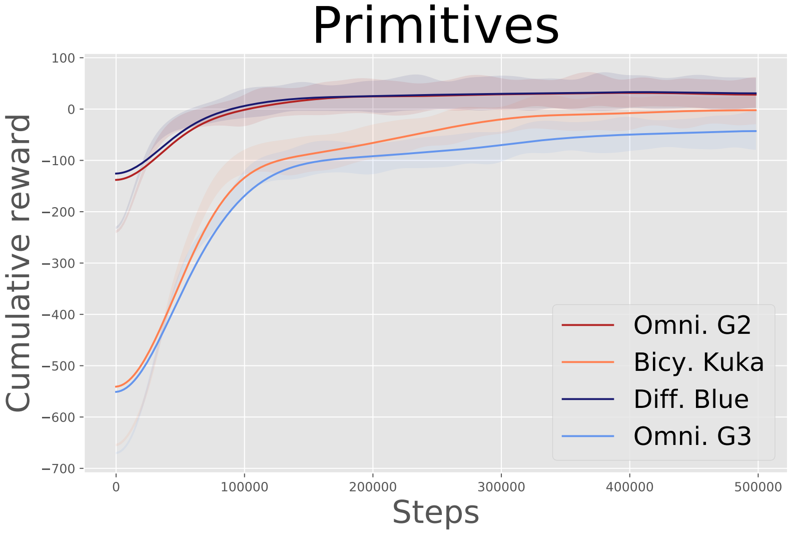


FIG. A.2. Training performances on Primitive reaching task for 4 predefined configurations

environment.

Curves in Figure A.2 are specific to our architecture and measure the output base training performances with respect to the reaching task. Figure A.3 and Figure A.4 respectively display these results for the whole architecture (bases + UNN) for the **Pick'n Place** and **Stacking** environments using an Omnidirectional Generic 2 robot configuration. Due to the intermediate model outputs, constrained through the bases, the UNN is able to learn the task more efficiently than a *naive* agent. As a matter of fact, we can see that both UNN-based techniques, in red and orange respectively representing the UNN and the BAM approaches, clearly outperform the *naive* PPO approach in blue. We note that the BAM (red) and UNN (orange) curves both reach a higher final cumulative reward than the *naive* PPO, while also being faster, indicating a better performance at solving the task overall.

Transfer

UNN-based approaches aim at making the skills transfer between agents of diverse constitutions possible. Hence, after training, we pass pre-trained models on a given configuration to another one with a different structure and compare their performances. In an ideal case, pairing the UNN would not require any fine-tuning. However, in practice, most cases require additional training iterations for the UNN to complete the adoption by another agent configuration. We display some examples of this process in Figure A.5 and Figure A.6, respectively for **Pick'n Place** and **Stacking** environments. These curves show for each environment how fast the performance can be recovered for a given configuration. Specifically, the y-axis measures the relative mean reward of the current agent projected in the lowest-to-highest interval from all initial agents. The x-axis also indicates how much of the initial training time was necessary to reach said performance. For the **Pick'n Place** environment, see Figure A.5, it appears that transfer within the *naive* framework is not efficient and even detrimental to the agent. The policy weights when transferred to another agent do not provide a decent initialization point and, through our experiments, these agents systematically failed at the task. On the contrary, using the UNN-based techniques allows to jump-start the new agent configuration with an initial level of performance that exceeds 70% of the previous agent and allows to recover almost the full performance, evaluated by the mean

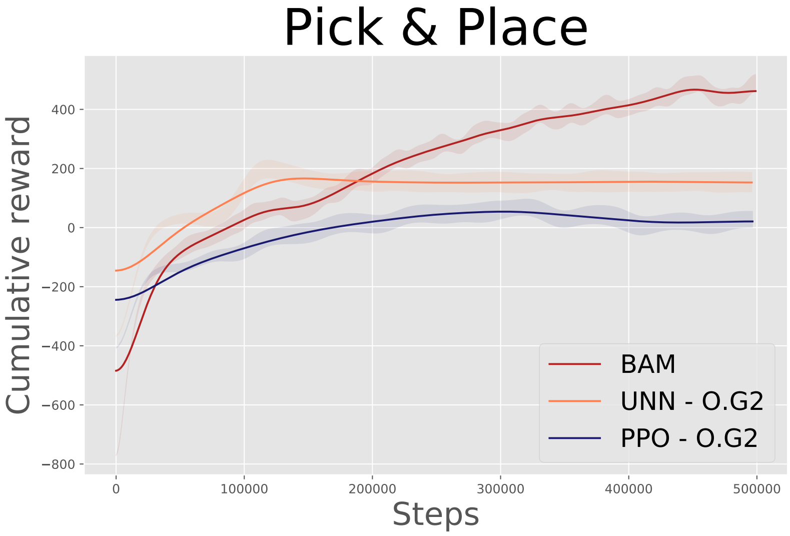


FIG. A.3. Comparative Pick'n Place training performances

cumulative reward per episode, in a fraction of the previous training time, mostly depending on the base conditioning. For the **Stacking** environment, similar observations emerge from the Figure A.6 where, again, the UNN-based methodology allows agents with different body configuration to access to the task knowledge, thus enabling quicker performances recovering.

While mean reward is a very common metric in RL, we also consider test time performances by averaging over 100 runs the number of steps necessary to complete the task for various agents in both environments. In this setting, the best agents are the fastest and need the lowest number of steps to complete the task. Table A.1 details results of this evaluation for **Pick'n Place** environment. In this case, we observe very contrasted results. No *naive* PPO agent was able to succeed in the task after being transferred, while the UNN-based transfer was particularly beneficial. Indeed, in these configurations, the performance was mostly conserved, with differences being related to the output base quality and movement capabilities (that is, it takes more time for the bicycle mobile base to reach a specific point than it does for the omnidirectional base.) This is further confirmed by results presented in Table A.2 concerning the **Stacking** task, where test performances are globally similar for all UNN-based agents with a transferred task-model. While these results are not sufficient to unequivocally determine an optimal configuration, they do however show that in both environments, the UNN and the BAM method yields an increase in performance and allow transfer between entities. This is due to the fact that, in this modelling approach, the UNN can focus on the task at hand, without having to learn simultaneously how to correlate external perceptions to low-level orders.

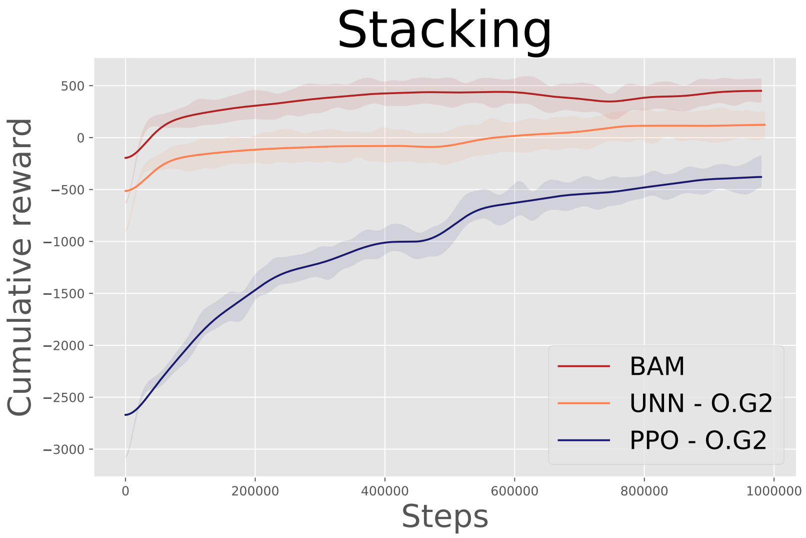


FIG. A.4. Comparative Stacking training performances

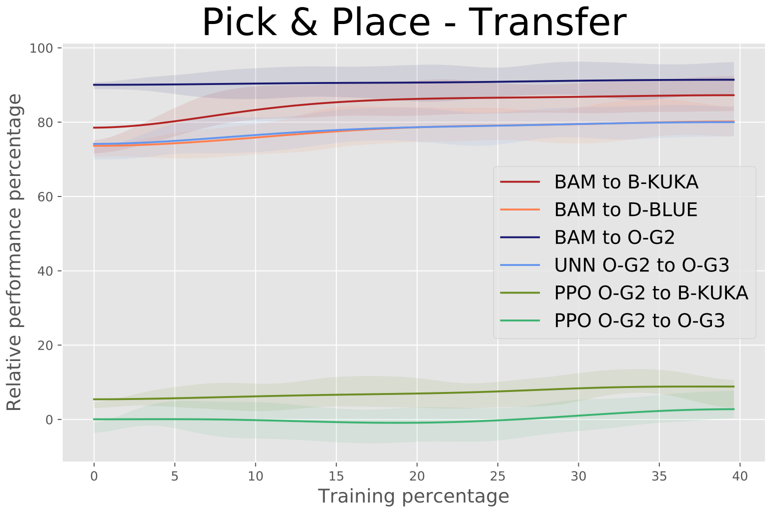


FIG. A.5. Pick'n Place transfer fine-tuning performances

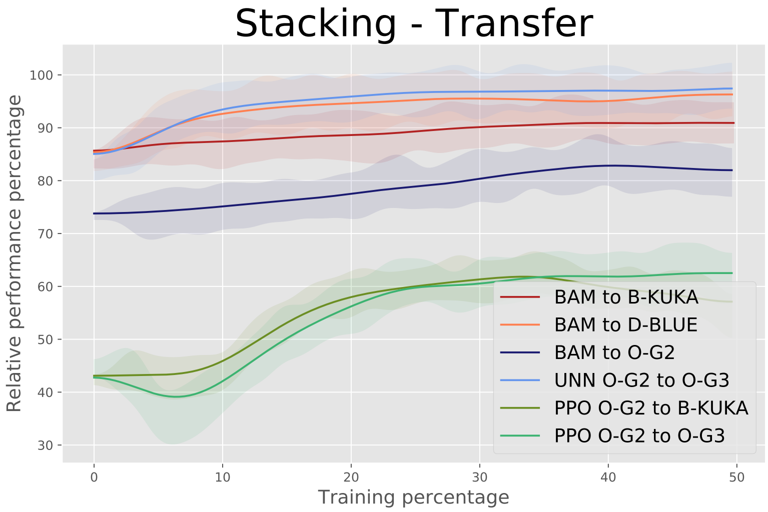


FIG. A.6. Stacking transfer fine-tuning

TABLE A.1. Transfer: Pick'n Place

Archi.	Config.	Agent	Training %	Perf.
	Initial	Omni. Generic 2	100%	1355.4
PPO		Omni. Generic 3	40%	Fail
	Transferred	Bicy. KUKA	40%	Fail
		Diff. BLUE	40%	Fail
	Initial	Omni. Generic 2	100%	848.8
UNN		Omni. Generic 3	20%	784.8
	Transferred	Bicy. KUKA	20%	801.7
		Diff. BLUE	20%	443.6
	Initial	-	100%	412.3
BAM		Omni. Generic 2	20%	751.6
	Transferred	Bicy. KUKA	20%	707.9
		Diff. BLUE	20%	747.7

TABLE A.2. Transfer: Stacking

Archi.	Config.	Agent	Training %	Perf.
PPO	Initial	Omni. Generic 2	100%	4571.9
	Transferred	Omni. Generic 3	40%	Fail
		Bicy. KUKA	40%	Fail
UNN	Initial	Omni. Generic 2	100%	3547.6
	Transferred	Omni. Generic 3	20%	4004.1
		Bicy. KUKA	20%	3966.1
		Diff. BLUE	20%	3847.9
BAM	Initial	-	100%	2879.3
	Transferred	Omni. Generic 2	20%	3540.4
		Bicy. KUKA	20%	3391.8
		Diff. BLUE	20%	3312.7