Vers un apprentissage protéiforme : accomoder les changements de signature des agents artificiels.

Iago Bonnici

▶ **To cite this version:**

Iago Bonnici. Vers un apprentissage protéiforme : accomoder les changements de signature des agents artificiels.. Intelligence artificielle [cs.AI]. Université Montpellier, 2021. Français. NNT : 2021MONTS034 . tel-03414933

HAL Id: tel-03414933
https://theses.hal.science/tel-03414933

Submitted on 4 Nov 2021

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

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

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École Doctorale : Information Structure Systèmes

Unité de Recherche : LIRMM

Vers un apprentissage protéiforme : accommoder les changements de signature des agents artificiels

Towards Protean Learning: Accommodating Signature Changes in Artificial Agents

Présentée par Iago BONNICI

Le 29 mars 2021

Sous la direction de Fabien MICHEL
et Abdelkader GOUAÏCH

Devant le jury composé de

Emmanuel RACHELSON, PR, ISAE-SUPAERO	**Rapporteur**
Matthew E. TAYLOR, Associate Professor, IRL Lab, University of Alberta	**Rapporteur**
Éric BOURREAU, MCF, LIRMM, Université de Montpellier	**Examinateur**
Antoine CORNUÉJOLS, PR, Dept. MMIP, AgroParisTech	**Examinateur**
Marianne HUCHARD, PR, LIRMM, Université de Montpellier	**Examinatrice**
Olivier SIGAUD, PR, ISIR, Université Pierre et Marie Curie	**Examinateur**
Abdelkader GOUAÏCH, MCF, LIRMM, Université de Montpellier	**Encadrant**
Fabien MICHEL, MCF HDR, LIRMM, Université de Montpellier	**Directeur**

et présidé par

Marianne HUCHARD, PR, LIRMM, Université de Montpellier	**Présidente du jury**

UNIVERSITÉ DE MONTPELLIER

Résumé :

Les agents de l'Apprentissage Automatique sont des procédures informatiques dont l'objectif est de découvrir, de façon systématique, des solutions à des tâches complexes comme la vision artificielle, le contrôle robotique, *etc.* Ce processus de recherche appelé "apprentissage" est caractérisé, comme tout programme, par un ensemble d'*entrées* possibles et un ensemble possible de *sorties* produites. Nous appelons ces ensembles la *signature* de l'agent apprenant.

Certaines tâches posent un défi particulier parce que la signature est naturellement amenée à *changer* au fil de l'apprentissage. Par exemple, un capteur qui cesse de fonctionner entraîne une "suppression d'entrée". Quand un nouvel actionneur devient disponible, il y a "ajout de sortie". À chaque changement de signature, l'espace des solutions exploré devient indéfini, et il est coûteux, en termes de calcul, de reprendre l'apprentissage de zéro. Dans cette thèse, nous abordons la question de l'accommodation du processus d'apprentissage aux changements de signature.

Rendre l'apprentissage robuste aux changements de contexte est un problème de longue date. L'Apprentissage Continu, la Dérive Conceptuelle ou l'Apprentissage par Transfert par exemple, améliorent sa robustesse dans des contextes évolutifs comme des distributions variables de données dans l'Apprentissage Supervisé Incrémental, ou des environnements fluctuants dans l'Apprentissage par Renforcement. Cependant, la question d'accommoder explicitement les changements de signature n'a pas encore, à notre connaissance, été abordée par la communauté. Nous défendons qu'elle représente un nouveau contexte d'apprentissage : l'Apprentissage Protéiforme (AP).

Dans un premier temps, nous établissons une formalisation rigoureuse de l'AP et du problème des changements de signature. En particulier, nous nous intéressons à l'ajout et la suppression d'entrée, qui constituent le sous-domaine restreint de l'AP d'Entrée (APE). Nous montrons qu'il existe 10 profils différents d'APE, et qu'un jeu de projections entre les espaces de solutions, dites *projections naturelles*, permet d'accommoder les évènements APE de façon générique.

Dans un second temps, nous concevons et réalisons deux expériences synthétiques pour mesurer l'avantage de ces projections dans des contextes diversifiés : descente de gradient "adam" dans un Réseau de Neurones Récurrent soumis à un tâche d'Apprentissage Supervisé Incrémental, "Q-Learning" et "Actor-Critic" avec traces d'éligibilité dans une tâche tabulaire d'Apprentissage par Renforcement. Dans chaque contexte, un agent protéiforme bénéficiant du transfert induit par les projections naturelles est soumis à une transformation élémentaire de signature en cours d'apprentissage. Ses performances à court et long termes sont comparées à celles d'un agent naïf reprenant l'apprentissage de zéro. Nous répliquons la procédure 1 000 fois dans chaque contexte, puis, par une analyse soigneuse des traces d'apprentissage, nous décomposons et documentons les nombreux effets en jeu pendant le processus d'accommodation APE, et établis par une *p-value* $\leqslant$ 0,01.

Nous montrons que les projections naturelles accommodent les changements de signature de façon plus économique que l'approche naïve, et nous discutons précisément des limites de cet avantage au regard des différentes propriétés contrôlées du contexte d'apprentissage, par exemple la difficulté de la tâche, l'informativité des senseurs gagnés ou perdus, la nature stochastique ou déterministe des stratégies recherchées, ou le profil APE. Nous concluons que l'intérêt de l'AP n'est pas seulement qu'il soulève une catégorie nouvelle de changements contextuels, mais aussi qu'il existe des techniques génériques, comme les projections APE naturelles que nous exhibons, qui les accommodent correctement dans les contextes testés.

Mots-clés :

Agents Apprenants, Apprentissage Protéiforme, Apprentissage Artificiel, Apprentissage par Transfert, Apprentissage par Renforcement, Apprentissage Incrémental, Réseaux de Neurones

Abstract:

Agents in Machine Learning are computer procedures aimed to automatically discover solutions to complex automation tasks like artificial vision, robot control, *etc*. This searching process called "learning" is characterized, like any computer procedure, by the set of possible *input* data they feed on, and the set of possible *output* data they produce. We refer to these sets as the learning agent *signature*.

Some learning tasks are challenging because the agent signature undergoes *changes* over the natural course of learning. When a sensor fails for instance, there is a sudden unexpected "input deletion". When a new actuator comes into use, there is an "output addition". But when the signature is transformed, the explored space of solutions becomes undefined, and restarting the learning from scratch is wasteful in terms of computing resources. In this thesis, we address the problem of accommodating signature changes in learning.

Making learning algorithms robust to changes is a longstanding problem. Approaches like Continual Learning, Concept Drift or Transfer Learning have long been concerned with the reaction of agents to evolving contexts, like varying data distributions in Online Supervised Learning or fluctuating environments in Reinforcement Learning. However, to our knowledge, the problem of explicitly accommodating signature changes has not been tackled by the community yet. We defend that it constitutes a significant new learning context, namely Protean Learning (PL).

First, we develop a rigorous formalization of PL and the signature change problem. In particular, we focus on input addition and deletion, which constitute the restricted Input-PL (IPL) subdomain. We show that there exists 10 different IPL profiles, and that a set of projections among solutions spaces, called *natural projections*, permits generic accommodation of IPL events

Second, we design and conduct two synthetic experiments to measure the advantage of natural projections in various contexts: an "adam" gradient descent in a Recurrent Neural Network solving an Online Supervised Learning task, "Q-Learning" and "Actor-Critic" approaches with eligibility traces in a tabular Reinforcement Learning task. In every context, one protean agent benefiting from the transfer induced by the natural projections is submitted to an elementary signature change event during the course of learning. Its short-term and long-term performances are compared against a naive agent restarting the learning from scratch. The process is replicated 1 000 times. Then, with a careful analysis of the learning traces, we dissect and document the numerous effects in play during IPL accommodation, and established with $p\text{-}value \leqslant 0.01$.

We show that natural projections permit better accommodation of the signature changes than the naive approach, and we qualify this advantage in depth depending on various controlled properties of the learning context, like the difficulty of the task at hand, the information contained in the input lost or gained, the stochastic or deterministic nature of the searched policy, or the IPL profile. We conclude that PL is not only interesting because it brings up a challenging new category of contextual changes, but also because there exist generic techniques, like the natural IPL projections we exhibit, that correctly address them in the tested situations.

Keywords:

Learning Agents, Protean Learning, Machine Learning, Transfer Learning, Reinforcement Learning, Online Learning, Neural Networks

Contents

Summary of Notations **7**

I Introduction 9

1 Introduction **11**
1.1 Context . . . 11
1.1.1 Signature Changes in Computing . . . 11
1.1.2 Machine Learning . . . 15
1.2 Learning Within a Changing Environment . . . 18
1.2.1 Signature of the Agent-Environment Retroaction Loop . . . 18
1.2.2 Reinforcement Learning . . . 21
1.2.2.1 A Trial and Error Search . . . 21
1.2.2.2 A Feedback Value to Guide the Search . . . 22
1.2.3 The Signature Change Problem . . . 24
1.2.3.1 Environmental Change . . . 25
1.2.3.2 Signature Change . . . 26
1.3 Thesis Outline . . . 29

2 Background: Machine Learning **33**
2.1 Behavioural Search . . . 33
2.1.1 The Maximization Problem . . . 34
2.1.2 Heuristics . . . 35
2.1.3 The Search Space . . . 38
2.1.4 Artificial Neural Networks . . . 39
2.2 Learning Contexts . . . 43
2.2.1 Supervised Learning . . . 44
2.2.1.1 The SL problem . . . 44
2.2.1.2 The SL Approach . . . 45
2.2.1.3 The Limits of SL . . . 46
2.2.2 Unsupervised Learning . . . 48
2.2.2.1 UL Principles . . . 48

2.2.2.2 Various UL Situations . . . 49
2.2.3 Reinforcement Learning . . . 51
2.2.3.1 RL Principles . . . 51
2.2.3.2 RL Methods and Limits . . . 53
2.2.4 Transversal Learning Contexts . . . 55
2.2.4.1 Online Learning . . . 55
2.2.4.2 Transfer Learning . . . 57

3 Problem Statement **61**
3.1 The Signature Change Problem . . . 61
3.2 Thesis . . . 62

4 State of the Art **65**
4.1 Documentation Method . . . 65
4.2 Related Works . . . 66
4.2.1 The Motivation for Transfer Learning . . . 66
4.2.2 The Various Transfer Learning Situations . . . 67
4.2.3 The Challenges of Transfer Learning . . . 70

II Contributions **73**

5 Theory of Protean Learning **75**
5.1 Informal Overview . . . 75
5.2 Formalization . . . 78
5.2.1 Background . . . 78
5.2.2 PL as a Problem of Stream Processing . . . 79
5.2.3 Data Streams and Causality . . . 80
5.2.4 Multiple Streams and Signatures . . . 83
5.2.5 Learning Dynamics . . . 85
5.2.6 The Objective of PL . . . 86
5.2.7 Discussion . . . 87
5.3 Input Addition and Deletion . . . 89
5.3.1 Time Dependency . . . 89
5.3.2 Diminished *vs.* Augmented Search Spaces . . . 90
5.3.3 Structure of the Search Spaces . . . 91
5.3.4 The IPL Landscape Profiles . . . 93
5.3.5 Using IPL Profiles . . . 95
5.3.6 The Natural IPL Projections . . . 97
5.3.7 Towards Generic IPL . . . 98

6 Experiment Input Protean Learning in Online Supervised Learning 101
6.1 Design . . . 101
6.2 Inputs Generation . . . 102
6.3 Optimal Outputs Generation . . . 104
6.4 The IPL profiles . . . 105
6.5 Agent Structure and Learning Procedure . . . 106
6.6 Signature Changes and Natural IPL Projections . . . 107
6.7 Measuring the Advantage of PL . . . 108
6.8 Results . . . 109

7 Experiment Input Protean Learning in Reinforcement Learning 117
7.1 A Tabular Benchmark for Input Protean Learning . . . 118
7.1.1 The transition function . . . 118
7.1.2 The Agent/Environment Interface . . . 120
7.1.3 A Minimal Environment . . . 122
7.1.4 Diminished Projections of the Environment . . . 123
7.1.5 Policy Types . . . 125
7.1.6 Selecting Transition Functions . . . 128
7.1.6.1 Transitions Connectedness . . . 129
7.1.6.2 Transitions Symmetries . . . 130
7.1.6.3 Sensitivity to Initial State . . . 131
7.1.7 Optimality Analysis of the Benchmark . . . 133
7.1.7.1 Computing Mean Rewards . . . 133
7.1.7.2 Optimal Deterministic *vs.* Optimal Stochastic Policies . . . 134
7.1.7.3 Joint IPL Profiles . . . 136
7.2 The RL Agents . . . 136
7.2.1 Explored Search Spaces . . . 137
7.2.2 The Agent Objective . . . 137
7.2.3 Q-Learning . . . 139
7.2.4 Actor-Critic . . . 142
7.3 Measures . . . 146
7.4 Results . . . 147
7.4.1 Last Performance Measures . . . 148
7.4.2 Long-Term Advantage Measures . . . 151
7.4.2.1 Input Addition . . . 152
7.4.2.2 Input Deletion . . . 153
7.4.3 Immediate Transfer Measures . . . 154
7.5 Discussion . . . 155

III Conclusion **161**

8 Conclusion **163**
8.1 Summary 163
8.2 Limits and Perspectives 165
8.3 Closing Thoughts 167

Annex **169**
A.1 Chapter 7 Transitions Function Symmetries 169
A.2 Constraints On Joint Profiles 174
A.3 Chapter 7 Additional Table and Figures 177

Bibliography **179**

List of Figures **194**

List of Tables **195**

List of Boxes **195**

List of Listings **195**

Following academic conventions, we use the first-person plural to refer to work done by the author, regardless of whether it was done in collaboration with colleagues. We also conform to gender-neutral language and use singular *they*, *their*, *themself*, to refer to a single person whose gender is unspecified.

Summary of Notations

General notations:

- $\varnothing$ The empty set.
- $\mathbb{N}$ The set of natural numbers.
- $\mathbb{Z}$ The set of relative numbers.
- $\mathbb{R}$ The set of real numbers.
- $\mathbb{R}^+$ The set of positive real numbers.
- $\mathbb{R}^{+*}$ $\mathbb{R}^+$ without the 0 element.
- $[0,\ 1]$ All numbers between 0 and 1 included.
- $\mathcal{U}n$ The uniform distribution.
- $\mathbb{P}(x \mid c)$ Probability that event x occurs given condition c.
- $\mathbb{E}(x \mid c)$ Expected value of random variable x given condition c.
- $\mathbb{1}_c$ Indicator function evaluated in c: 0 if c is false and 1 if c is true.
- $|\mathcal{X}|$ The number of elements in set $\mathcal{X}$, a.k.a. the cardinal of $\mathcal{X}$.

Whenever possible and non-ambiguous, capital letters (A, P, F, *etc.*) are used for procedures, functions and distributions. calligraphy letters ($\mathcal{S}$, $\mathcal{I}$, $\mathcal{O}$, *etc.*) for sets and lowercase letters (s, i, o, *etc.*) for values, random variables and and elements living in these sets:

- $F : \mathcal{X} \to \mathcal{Y}$ The function F maps set $\mathcal{X}$ into set $\mathcal{Y}$.
- $F : x \mapsto y$ The function F maps value x to value y.
- $y = F(x)$ The value y is the result of function F applied to value x.
- $y \sim F(x)$ The random variable y is distributed according to the probability distribution $F(x)$ parametrized by the value x.
- $x \multimap (F) \rightarrow y$ The data stream y is determined from the data stream x by F in a causal, non-Markovian way (see Chapter 5).
- $F^* \triangleright \mathcal{X}$ The value F^* maximizes F over $\mathcal{X}$, or $F^* = \max_{x \in \mathcal{X}}(F(x))$.
- $\mathcal{X} \rightarrowtail \mathcal{Y}$ There exists an injection from set $\mathcal{X}$ to set $\mathcal{Y}$.
- $\mathcal{X} \twoheadrightarrow \mathcal{Y}$ There exists a surjection from set $\mathcal{X}$ to set $\mathcal{Y}$.
- $\mathcal{X} \leftrightarrow \mathcal{Y}$ There exists a bijection between set $\mathcal{X}$ and set $\mathcal{Y}$.

The following symbols are used pervasively to represent particular learning concepts, unless otherwise stated. They are also used outside strict mathematical contexts to refer to the concept in general and not to one precise mathematical object:

- A Learning agent procedure.
- E Agent environment.
- T Transition function.
- P Inner procedure, or "policy" of the agent, responsible for its end behaviour (not to be confused with probability symbol $\mathbb{P}$).
- $\mathscr{P}$ Search space explored by the agent during learning. $\mathscr{P}$ contains every considered procedure $P \in \mathscr{P}$.
- S Inner search procedure supposed to explore $\mathscr{P}$.
- i Inputs to the learning agent and the behavioural procedure.
- o Outputs from the behavioural procedure and the learning agent.
- f Feedbacks to the learning agent procedure: rewards and penalties.
- s Elementary environmental state.
- h Hidden state.
- u Elementary control, or agent action.
- $\mathscr{E}$ Set of possible environments.
- $\mathscr{T}$ Set of possible transition functions.
- $\mathscr{I}$ Set of possible input values.
- $\mathscr{O}$ Set of possible output values.
- $\mathscr{F}$ Set of possible feedback values.
- $\mathscr{S}$ Set of possible environmental states.
- $\mathscr{U}$ Set of possible elementary agent actions.
- I Observation procedure, converting states to inputs.
- O Actuation procedure, converting outputs to actions.
- F Feedback procedure, objective function.
- Δ The agent signature, or arbitrary domain of values.
- ℓ The natural projection from diminished landscapes to augmented landscapes.
- $\wp$ An almost-natural projection from an augmented landscape to a diminished landscape.
- m The natural injection from the space of deterministic procedures to the space of stochastic procedures.

Part I

Introduction

Chapter 1

Introduction

1.1 Context

1.1.1 Signature Changes in Computing

Computer science, as a scientific discipline, is characterised by numerous extensive application domains that differ much in nature: computation, simulation, communication, creation, entertainment, control, problem solving, *etc.* At their core, there is *computation*. Computation is a process that transforms elementary inputs into elementary outputs by strict application of a predefined procedure. For instance, the addition procedure is an operation that transforms, say, a binary representation of the numbers 1 and 2 (0001 and 0010) into the corresponding representation of 3 (0011). Let us represent this process as:

$$\textit{input}\ \textit{[0001 and 0010]} \rightarrow \textit{procedure}\ \textit{[addition]} \rightarrow \textit{output}\ \textit{[0011]}$$

Or to make it shorter with an informal diagram:

$$i \rightarrow P \rightarrow o$$

Machines perform billions of elementary operations within seconds without mistaking or getting tired. As such, they are much more suited than humans to execute such repetitive and well-defined computation procedures: boolean operations, integer arithmetic, floating point calculations, text processing, *etc.* Computation arguably roots all other applications of computer science, because computer-assisted simulation, communication, creation, entertainment, control, problem solving *etc.* all rely on these elementary operations being correctly and efficiently executed *in fine*.

Some applications are straightforward extensions of computation. In *simulation* for instance, the user has determined a rigorous set of evolution rules P, and one particular input used as a starting point. To perform the simulation, an output is computed from the initial input, then it is immediately reused as a new input in the next simulation step to get another output, and so on.

For instance, a weather prediction model is a simulation constituted of a set of complex evolution rules representing spontaneous evolution of air pressure, temperature, hygrometry *etc.* If we have weather measures taken in the last hour, we feed them as an initial input to the model, and it outputs a weather forecast for the next hour. This forecast can be used as an input again to get a prediction two hours ahead, *etc.* until we are satisfied, or until the predictions get too weak. Under this view, simulation is just computation iterated over time, like:

$$\begin{aligned} &\textit{input}\ \textit{[latest weather measures]} \rightarrow\ \textit{procedure}\ \textit{[weather model]} \rightarrow \textit{output}\ \textit{[1-hour forecast]} \\ &\quad \rightarrow \textit{input}\ \textit{[1-hour forecast]} \rightarrow \textit{procedure}\ \textit{[weather model]}\ \rightarrow \textit{output}\ \textit{[2-hours forecast]} \\ &\quad \rightarrow \textit{input}\ \textit{[2-hours forecast]} \rightarrow \ldots \end{aligned}$$

Or to make it shorter:

$$i \rightarrow P \rightarrow o \rightarrow i \rightarrow P \rightarrow o \rightarrow \ldots$$

Arguably, all the above simulation process can be rewritten as one single, automated, iterative procedure $\boldsymbol{P_{it}}$. This single procedure feeds from the same initial input, but it produces the whole sequence of predictions at once, in a single output:

$$i\ \textit{[latest weather measures]} \rightarrow \boldsymbol{P_{it}}\ \textit{[repeated calls to P]} \rightarrow o\ \textit{[forecasts for the next few hours]}$$

Note that this single procedure is now iterative and executes inner procedures itself. In this situation, the caller of the inner procedure P is not a human user, but the higher-level procedure $\boldsymbol{P_{it}}$ instead. Nesting procedures into each other this way, and defining new procedures from existing ones, is the natural way humans use machines and build up sophisticated computer processes from elementary computational operations.

This makes the notion of *procedure P* akin to the notion of mathematical *functions* and to the notion of computer *programs*, that nest into each other. They are the cornerstone of computer science, and they have 3 obvious ends:

- i: The *input* data, used to feed the procedure. Without inputs, no computation is made.
- o: The *output*, or outcomes of the process. They are typically data, but they also constitute real-world effects like motor activation or pausing. Without outputs, the computation is useless.
- P: The *procedure* itself, responsible for execution of the process. Without a procedure, it is unspecified how the output is supposed to be produced.

Inputs and outputs are easy to describe. In computer science, they are most commonly *data* whose size, shape and meaning are understood both by the caller and by the procedure. On contemporary machines, they are typically sequences of bits representing numbers, text, pictures, speech, *etc.* For instance, the `addition` procedure defined earlier expects two 4-bits-long sequences representing integers (*e.g.*, 1 and 2), and produces one

4-bits-long sequence also representing an integer (*e.g.*, 3). Higher-level procedures like the weather prediction model expect more sophisticated data structures. In this situation, inputs represent incomplete spatialized atmospheric properties, and outputs are similar structures associated with confidence estimations. Alternate possible kinds of inputs include knowledge graphs, other agents information, books *etc.*

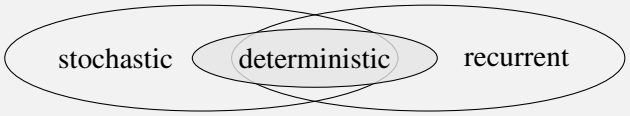
The shape and meaning of input and output data constitute the very essence of the interplay between the procedure and its caller, and in particular, between humans and machines. We refer to them as the procedure *signature*. The signature is a rigorous contract between both members. On the one hand, the procedure is expecting that the input data be well-formed and meaningful, and is responsible for producing correct output. On the other hand, the caller is expecting that the output data be also well-formed and meaningful, so it is responsible for feeding correct input in. As a consequence, the input/output signature cannot be changed unless both members are aware and update accordingly.

The present thesis addresses *signature changes* in one particular class of computer procedures called *learning procedures*. Learning procedures are extensively described in the remainder of this introduction. Regarding signature changes, in general, there are several possible reasons for a procedure signature to change. For instance, the procedure designer decides to change the signature when they realize that it does not exactly fit their objectives, or they have new ones. As an example, the weather prediction model improves after upgrade when it also outputs holidays traffic jams predictions in addition to its own forecasts. This is a case of *output addition*. Alternately, the data processed by the procedure occasionally becomes obsolete, or unavailable. For instance, when the hygrometers fail or the wind records from last week are lost due to a server crash, the internal procedure of the weather prediction model needs to be adjusted to accommodate the missing information. This is a case of *input deletion*. There exist various types of signature change events, which are the object of our work. In the second part of the thesis, we focus specifically on *input addition* and *input deletion*. As written above, we address in particular how one special class of computer procedures, the *learning procedures*, react when such an event occurs.

To understand what learning procedures are, we first need to describe computer procedures more in depth. This is more difficult than describing inputs and outputs, because a computer procedure is a deep, complex, dynamical and flexible object. Most often, it is defined recursively by inner calls to lower-level procedures, until nothing but physical laws actually specify what the output is. For instance, the addition procedure results from combinations of bit-level logical AND and XOR procedures, whose efficiency and correctness is only guaranteed *in fine* by the electronics of computer chips. In a nutshell, every computer procedure, including weather forecast, is ultimately defined, and constituted, by carefully triggered natural processes.

This has three very important consequences. First, unlike mathematical functions, it is mandatory that computer procedures be *constructive* and *finite*. No computer can out-

Box 1.1: Non-deterministic procedures.



Mathematical functions are strictly *deterministic*. This means that, given twice the same input i, they always output the same value $o = P(i)$. Computer procedures, on the other hand, are not mandatorily deterministic. For instance, computer output cannot be guessed when it generates user passwords at random. Procedures relying on random processes are called *stochastic* procedures. They are specified in terms of probability distributions, *e.g.*, $o \sim P(i)$, and are useful in simulation, games, cryptography *etc.* But there exists other sources of non-determinism. For instance, some computer procedures are called *recurrent* because they keep track of previous executions and modify their current output depending on past events. This is the case of user login creation procedures, which output "ok" but then "already in use" if the same username is input twice. This implies that recurrent procedures feature a *memory* of past events, sometimes referred to as inner *hidden states* h. A typical specification of recurrent procedures involves additional, hidden inputs and outputs, inaccessible to user: *e.g.*, $(o,\ o_h) = P(i,\ i_h)$. This is useful in sequential decision making, modelling natural processes, or Reinforcement Learning (see Chapter 2). Obviously, various sources of non-determinism can be combined together, *e.g.*, $(o,\ o_h) \sim P(i,\ i_h)$. There also exists other sources of non-determinism than the ones just described, like *e.g.*, concurrency, undefined behaviour [J. Lee *et al.* 2017], *etc.* As drawn above, it is convenient to consider that deterministic procedures constitute a degenerated case of non-determinism that belongs to every other category: *e.g.*, deterministic procedures are stochastic procedures with only 0 or 1 probability measures, and also recurrent procedures with empty hidden states.

put a result that no human could effectively compute by themself with natural processes, provided they have enough resource to do so. Second, just like the natural processes they eventually rely on, computer procedures are not necessarily *deterministic*. They can also exhibit any kind of non-deterministic behaviour like *stochastic* behaviours or *recurrence* (see Box 1.1). Third, and just like mathematical functions, computer procedures are ultimately defined by *humans*, and machines are only able to execute them.

The latter seems intuitive when it comes to straightforward applications of computer science, like plain computation, approximation, simulation, communication, *etc.* because they suggest sophisticated combinations of recursive procedures, designed and written by humans with a particular purpose in mind. In these domains, the challenge is to have them correctly executed by the machine, preferably in an effective way in terms of computational "resources": computation time, memory usage, disk storage, energy consumption, development time, investor money, *etc.*

However, it seems conflicting with other applications of computer science. Most famously, no human plays go better than Ke Jie, yet he has been beaten by a machine in 2017. Also, no human knows a constructive procedure to perform face-recognition on random internet pictures, yet some clever, neural-networks-based algorithms learn doing this "on their own". In achieving these goals, computer scientists face a problem where the expected inputs and outputs are easily described, (*e.g.*, i: location of every stone on the go board, o: next winning move), but the procedure P that would correctly connect them is so difficult to design that it is out of practical reach. Artificial Intelligence (AI) is concerned with this very situation, and attempts to exploit the computational power of machines to bypass these human limitations.

This is were the *learning procedures*, the particular category of procedures whose reactions to signature changes we are studying in this work, come into play. The next section defines what learning procedures are, along with the associated research domains. Section 1.2, illustrates the signature change problem with an example learning procedure.

1.1.2 Machine Learning

There are problems that humans are able to solve, but it is very hard to design a computer procedure also able to solve them. Put it another way, there are "elusive procedures", $P^\star$. Among the most common ones: recognizing objects in our immediate environment, understanding questions asked by another human, producing adequate answers in natural language, or issuing right decisions in governance debates. These elusive procedures seem to form a gap between humans and machines, that the domain of AI has been historically willing to bridge. The question whether it is possible to bridge it belongs to philosophy, as it essentially boils down to the deep nature of human beings: If machines are limited by natural processes, are we? And if we are, why could they not succeed in anything we do? Notwithstanding philosophy, the AI community as a whole agrees that designing a computer procedure $P^\star$ able to perform the above tasks correctly and efficiently is a hard problem. This is mostly because we lack knowledge about the way humans actually succeed in these. This knowledge belongs to the fields of cognitive sciences, of psychology and sociology, and is yet unveiled.

The AI problem is traditionally tackled by researchers from a symbolic, logical perspective. The idea is that the elusive procedures $P^\star$, if they exist, must eventually rely on the basic rules of human reasoning, a.k.a. fundamental logic. This "strong" AI approach is ambitious as it focuses on the difficult question: *"How should $P^\star$ procedures work?"*. It struggles with concerns that eventually relate to the unknown nature of the human mind.

Alternately, for a few decades, there has been growing interest in new "weaker", more pragmatic approaches, because they turn out to be surprisingly powerful. In the domain of Machine Learning (ML), the idea is that the elusive procedures $P^\star$ be automatically found by artificial, automated agents dedicated to particular tasks. As a consequence, these approaches only need to answer the question *"What should $P^\star$ procedures do?"* [Ring 1997]

and the expected behaviour can be fulfilled by approximated procedures that yield acceptable performances. In other terms, ML is an empirical, statistical approach to AI. In particular, ML agents only need to know the *signature* of the elusive procedures $P^\star$ to run, and not the procedures themselves. This is the reason why signature changes are particularly interesting to address in this context.

ML is successful because the agents it produces are typically generic in theory, in the sense that they are supposed to succeed in various different tasks, and still are both useful and efficient in practice. Artificial ML agents do recognize objects in pictures, understand partial human speech, beat humans at board games, or again, predict online consumers behaviours, whereas no known human is able to design the corresponding formal procedures $P^\star$. This seems to contradict with the computer procedures limitations defined earlier. If anything a machine does is to follow predefined procedures, how can these algorithms succeed in tasks for which no explicit procedures were written?

The trick of ML is to not even try designing the elusive procedure $P^\star$. Instead, ML defines a wide set of possible, well-defined procedures $\mathscr{P}$, and assumes that this set is broad enough to contain at least one procedure $\hat{P}$ that approximately resembles $P^\star$. Once this has been set, the idea is to search $\mathscr{P}$, find $\hat{P}$, and be satisfied with it.

There are two major practical problems in achieving this. First, there is no way to tell which procedures in $\mathscr{P}$ are the right ones, so ML typically needs to successively test them in turn until they find $\hat{P}$. Second, it is very common for the search space $\mathscr{P}$ to be immense or infinite, so the task of searching it is much resource-consuming. Fortunately, searching $\mathscr{P}$ constitutes a well-defined repetitive procedure, which, as we wrote earlier, is something machines are particularly successful at.

In a nutshell, the key insight of ML is to use the computational power of machines to make them search the space of possible procedures $\mathscr{P}$ by themselves, until they find a satisfying approximation of $P^\star$. To this end, artificial agents A are constructed, and they follow *searching procedures*. Searching procedures perform similarly to the weather simulation model defined earlier. For instance, on each time step, A receives one candidate procedure P from $\mathscr{P}$ as an input, calls it to evaluate it, and outputs whether or not it is likely to be useful, along with the next procedure to be tested:

$$\begin{aligned} & i \textit{ [first candidate P]} \rightarrow A \textit{ [test P]} \rightarrow o \textit{ [quality of P and second candidate]} \\ \rightarrow\ & i \textit{ [second candidate P]} \rightarrow A \textit{ [test P]} \rightarrow o \textit{ [quality of P and third candidate]} \\ \rightarrow\ & i \textit{ [third candidate P]} \rightarrow \dots \end{aligned}$$

In the above process, all procedures A and P are predefined and designed by humans, so it does not contradict with the fundamental limitations of machines described earlier. This automated procedure search eventually yields a good quality procedure $\hat{P}$.

Note that this procedure is not $P^\star$. In fact, it is not even expected by computer scientists that $\hat{P} = P^\star$, because the chances that $\mathscr{P}$ actually contains $P^\star$ *and* that A finds it exactly are insignificant. Moreover, $\mathscr{P}$ typically contains procedures dedicated to approxi-

Box 1.2: Epistemological status of ML.

In spite of a loud success, ML suffers from an ambiguous reputation. On the one hand, its effective performances in numerous applications are exciting because they seem to seamlessly "bridge the AI gap". On the other hand, its pragmatic statistical nature unveils very few fundamental knowledge about the "elusive procedures" that were the object of "strong" AI. In other terms, ML explicitly rules out $P^\star$ to focus on performance with approximated $\hat{P}$. It produces procedures that mimic the ones we expect, but teaches us nothing about them. This is a poor epistemological perspective. The disappointment is magnified by numerous misleading terms commonly used in ML like "learning" (searching $\mathscr{P}$), "neural networks" (one tool to scroll candidates P) or "knowledge" (the final procedure $\hat{P}$) [Wladawsky-Berger 2020]. These terms suggest a close relation between ML procedures and the human mind, but they are fallacious. The relation is a regular human/machine interaction like any computation: Humans define procedures with a particular purpose in mind (search $\mathscr{P}$ for $\hat{P}$) and machines execute them. In this thesis though, we still consider that this approach has much epistemological interest, especially at its frontiers. As a matter of fact, ML does not solve *every* problem out of the box. Some elusive procedures $P^\star$, (autonomous robot control, language acquisition) are still far out of practical reach even with the ML trick and today powerful machines. "Solved" tasks, like playing chess or recognizing pictures, also carry their load of unresolved questions. For example, we understand that any $\hat{P}$ is satisfying provided it is close enough from $P^\star$, but it is unclear what makes one particular $\hat{P}$ useful. $\hat{P}$ is just one point in a wide space of possible procedures $\mathscr{P}$, why does it behave so closely to humans? Does this reveal anything about the human mind? Do we learn by exploring, like formal procedures, an environment we know nothing about, and a space of possible behaviours so wide that it puzzles humans and machines alike? The reason that misleading terms are used in ML is that ML actually roots in bio-inspiration and Artificial Life. Neural Networks are loosely inspired from the human brain [McCulloch and Pitts 1943]. Reinforcement Learning is inspired from Psychology [Sutton and Barto 2018]. Automation of natural speech processing highlights subtle linguistic phenomena [Goldberg 2016]. In the end, ML and cognitive sciences, psychology, sociology, are not entirely distinct as they keep interacting together in search for the mechanics of the human mind. Any progress in one unlocks progress in the others. And there is room for improvement in many practical and theoretical aspects, like the *signature change* problem addressed in this work.

mation, and not supposed to resemble human models of the problem, so implementations of $\hat{P}$ and $P^\star$ differ much by design. This said, $\hat{P}$ is good enough to yield astonishing results like synthesizing human faces [Karras *et al.* 2018] or beating human chess champions [Hsu 2002], which heightens the attention given to automated searching procedures.

Another common term for the searching procedure is a *learning procedure*, because it is said that the agent "learns" how to solve the task at hand (see Box 1.2). In this thesis,

we investigate how learning procedures accommodate *signature changes*. In particular, signature changes destroy and redefine the procedures search space $\mathscr{P}$ during the learning process, and restarting the search from scratch is inefficient. Can better approaches be developed to accommodate these events?

1.2 Learning Within a Changing Environment

In this thesis, we are interested in the problem of *signature changes* in learning computer procedures. As described in the previous section, signature changes constitute a natural phenomenon that affects procedures subject to variations in their basic input/output contract. To our knowledge, this phenomenon is not accurately captured by modern ML approaches, and very few techniques accommodate it when it occurs during the course of learning.

This section describes the phenomenon more in depth, with the help of an illustrative example (a fictitious roverbot) that we refer to until the end of the chapter, and then regularly throughout the manuscript. Before we expound signature changes, we need to detail the interface between an agent subject to signature changes and its environment. This is the object of the next section.

We take this opportunity to introduce Reinforcement Learning (RL) from an original, stream-oriented perspective. This perspective is reused throughout the manuscript to integrate signature changes as seamless events within the data streams. In particular, we use it to construct the formal model defined in Chapter 5. We also choose to stand off the usual notations in RL (i, o, P, E, *etc.* instead of s, a, π, T, *etc.*) to make it explicit that the concepts developed in this thesis are not restricted to RL, but generic to various domains in ML as explained in Chapter 2. In Chapter 5, we discuss the connections between our notations and the traditional ones.

1.2.1 Signature of the Agent-Environment Retroaction Loop

Consider a sticky roverbot whose task is to follow a user anywhere (see Figure 1.1-Left). The robot is simple and imaginary, as it is only used to introduce the key concepts of the thesis. It is equipped with a few *sensors* providing the basic *inputs*, i, needed to fulfill the task, *e.g.*:

- A front camera to spot the user direction.
- A front laser to measure distance to user.

It is also equipped with a few *actuators* providing the basic capabilities required to fulfill the task, *e.g.*:

- A steering bar, associated with a motor controlling wheels direction.
- A power switch, that controls whether the wheels are spinning frontwards or backwards, or not.

Those are considered the *output*, o, of the robot.

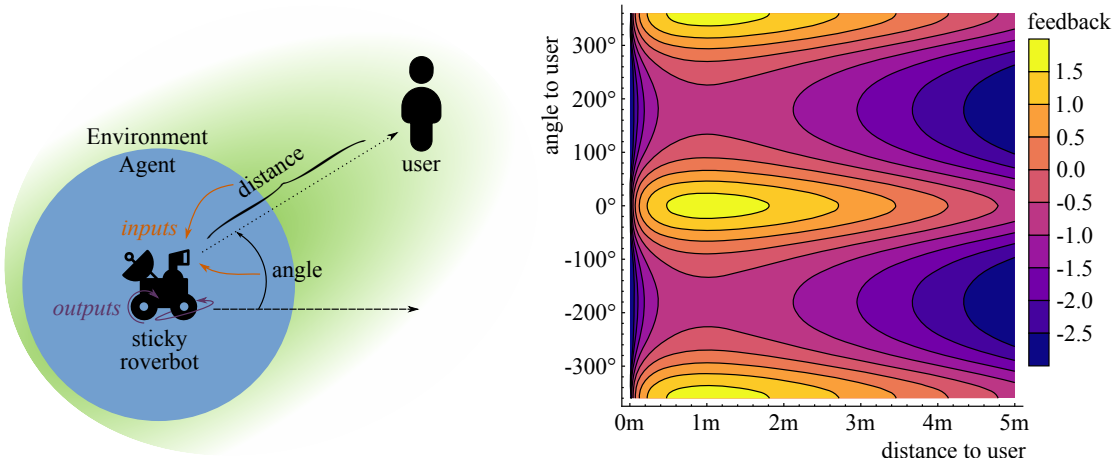


Figure 1.1: Left: The imaginary sticky roverbot used as an example learning agent in this chapter. Right: Contour plot of an example feedback profile for the roverbot, defined with respect to the user distance and angle.

Naturally, the robot also contains a chip that is responsible for hosting and running its inner computing *procedure*, P. The robot procedure is supposed to choose the right actions to undertake given the situation currently perceived by the sensors. *In fine*, P determines the robot end behaviour. The robot *signature* is the type of data supposed to flow through P: angle, distance, steering direction, power switch state. The signature can be thought of as the list of sensors and actuators currently featured by the roverbot.

This situation raises several scientific challenges, most of which relate to the roverbot engineering itself: power supply, wiring, robustness, flexibility, motion, safety, *etc.* In particular, it also raises computer science related challenges. For instance, how to compute user direction from the front camera picture? Or how to correctly coordinate the motors so they do not break mechanical constraints? In the domain of ML, there is focus on one particular question: What to use as the robot inner procedure P? In this thesis, we focus on the associated question: How can P accommodate signature changes?

Within the roverbot, P is not a short, one-shot procedure like the addition procedure seen earlier. With the addition procedure, you feed inputs in once (*e.g.*, 9 and 3: 1001 and 0011), then you get the output once (*e.g.*, 12: 1100) then the process is over:

$$i \to P \to o$$

The roverbot procedure is different because it lives *ongoing* time. On each time step (say, every second), new decisions need to be taken depending on latest inputs. This resembles the weather *simulation* procedure seen earlier, where each output was immediately reused as a new input to keep predicting a few hours ahead:

$$i \to P \to o \to i \to P \to o \to i \ldots$$

In the weather simulation case, the whole process was only determined by i and P so we could easily rewrite the above as a higher-level one-shot procedure $\boldsymbol{P_{it}}$ that did the whole prediction at once:

$$i \textit{ [initial weather measures]} \rightarrow \boldsymbol{P_{it}} \textit{ [inner calls to P]} \rightarrow o \textit{ [e.g., 10 hours weather predictions]}$$

But with the roverbot, the new inputs available on a time step are *not* the result of previous computation. Instead, they constitute new, fresh information taken from the outside world, *i.e.* the agent's *environment* E:

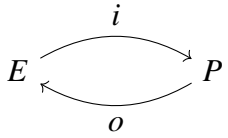
$$E \rightarrow i \textit{ [user at 28°, 3.31 m]} \rightarrow P \rightarrow o \textit{ [stop running, steer]}$$
$$E \rightarrow i \textit{ [user at 10°, 3.30 m]} \rightarrow P \rightarrow o \textit{ [keep steering, but slower]}$$
$$E \rightarrow i \textit{ [user at -2°, 3.30 m]} \rightarrow P \rightarrow o \textit{ [stop steering, run]}$$
$$E \rightarrow i \textit{ [user at -1°, 3.16 m]} \rightarrow \ldots$$

In the general case, the dynamics of the environment E are unknown from both the procedure P and from the bot designer themself. They encompass complex and unpredictable phenomena like user behaviour, ground quality, weather conditions, random fluctuations of sensors, *etc.* As a consequence, the above process cannot be rewritten ahead of time as a one-shot summarized procedure $\boldsymbol{P_{it}}$ like in the previous situation. The roverbot has to keep repeating a fresh decision process P indefinitely to keep running.

Interestingly, although the environment behaviour is ultimately unpredictable, there is much that the agent can do to make it behave one particular way. For instance, if the bot uses steering to face the user, then rolls towards them, there are good chances that the distance to user eventually decreases down to only a few centimeters. Of course, unexpected things happen if the user keeps moving around, if there are obstacles on the ground, or enough wind to deviate the bot trajectory. But the agent decisions constitute a legitimate share of the set of factors determining the values of next inputs. In simpler terms, the agent outputs influence the environment, like in:

$$E \rightarrow i \rightarrow P \rightarrow o \rightarrow E \rightarrow i \rightarrow P \rightarrow o \rightarrow E \rightarrow i \rightarrow \ldots$$

Or, to make this agent-environment retroaction loop more explicit:



In this view, E and P form a dynamical system where each procedure influences the other *via* the only communication channels of the agent: i and o. This makes it very clear that the agent *signature* is essential to the integration of P within its environment. Computer scientists have very low control over E because it involves natural physics, user behaviour or even chance. But they are supposed to design P and its signature so

that the roverbot fulfills the task no matter the possible hurdles in E. This is a difficult problem, as it relates to AI, and the ideal P can be considered an "elusive procedure" $P^\star$.

Note: The "$\rightarrow$" arrows used in this chapter are drawn for illustrative purpose only. In Chapter 5, a formal definition of " $\multimap\!(\,)\!\longrightarrow$ " arrows makes it possible to carry rigorous meaning with similar diagrams.

Sometimes, the signature of the roverbot changes as the result of natural phenomena. This happens for instance on an sensor failure (*e.g.*, the laser breaks) or actuator addition (*e.g.*, a new speaker is plugged in). When this happens, the procedure needs to accommodate the change. Within ML, there exists learning procedures able to address agent-environment retroaction loop contexts like the one described above. These procedures are very common and well-documented but, to the best of our knowledge, they do not yet accommodate signature changes. We describe one such procedure in the next section.

1.2.2 Reinforcement Learning

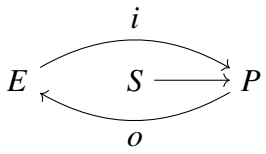
As explained in Section 1.1.2, the ML approach to the problem of animating the roverbot is not to try to write the elusive procedure $P^\star$ explicitly. Instead, a wide space of possible procedures is designed: $\mathscr{P}$, and the new agent task is to search this space until it finds a procedure that makes it approximately follow the user in acceptable fashion: $\hat{P}$. ML branches into several subdomains depending on the learning situation involved (see Section 2.2). The roverbot situation described here is most naturally handled by one particular ML approach called Reinforcement Learning (RL) [Sutton and Barto 2018]. RL is typically useful when the procedure to learn is involved in an ongoing agent-environment retroaction loop like the one above. This section introduces the basic principles of RL needed to expose the thesis, RL is developed more in depth in Section 2.2.3.

1.2.2.1 A Trial and Error Search

When using RL to solve the roverbot problem, the first important thing to consider is that the bot inner procedure P is not fixed anymore. Instead, the agent tries several candidate procedures P in turn, all drawn from $\mathscr{P}$, until it finds $\hat{P}$. The agent "search" procedure S is responsible for performing this search:

$$
\begin{aligned}
&S \textit{ [pick from } \mathscr{P} \textit{]} \rightarrow P \textit{ [first candidate]} \\
&\qquad E \rightarrow i \rightarrow P \textit{ [first candidate]} \rightarrow o \rightarrow \dots \\
&S \textit{ [pick from } \mathscr{P} \textit{]} \rightarrow P \textit{ [second candidate]} \\
&\dots \rightarrow E \rightarrow i \rightarrow P \textit{ [second candidate]} \rightarrow o \rightarrow \dots \\
&S \textit{ [pick from } \mathscr{P} \textit{]} \rightarrow P \textit{ [third candidate]} \\
&\qquad \dots
\end{aligned}
$$

Drawn another way:



This has an important immediate consequence: before it finds $\hat{P}$, the roverbot undergoes many different behaviours outputted by S, some of which do not fulfill the task at all (*e.g.*, not moving, roaming in circles, flee from the user, flip upside down). During this uncertain "learning phase", the robot must be able to safely perform such *trial and error* runs, until it eventually converges towards an acceptable behaviour $\hat{P}$. This "heuristic" disadvantage is the cost paid by ML computer scientists to not have to write the elusive procedure $P^\star$ explicitly. This cost is smoothly endorsed for theoretical, imaginary agents like the one at hand, but undesirable when it involves a real-world robot either precious, dangerous or fragile. As such, RL is barely used in robotics in practice, but see [Smart and Pack Kaelbling 2002; Bakker *et al.* 2006; Hester and Stone 2013; Cutler and How 2015] for insights into bridging this gap. Overcoming the cost is easier with virtual agents, like software tracking user preferences, because they cannot severely harm themselves or their immediate environment. Still, the learning phase is typically expensive in terms of computer time, data consumption, *etc.* As such, any technique susceptible to reduce the cost of the learning phase is considered welcome in the domain of ML.

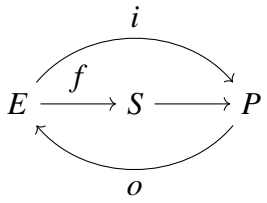
This permits a better characterization of the signature change problem. When the roverbot signature changes, the search space $\mathscr{P}$ is also transformed, so it needs to be explored again to find a new $\hat{P}$. One trivial solution to accommodate the change is to start the search over. Therefore, RL agents easily undergo such events by just resuming the learning from scratch and exploring the new $\mathscr{P}$ again. Unfortunately, this naive technique comes at the cost that the whole learning phase needs be undergone again. In this thesis, we investigate alternate techniques so that parts of the "knowledge" collected by the agent in the former search space be reused in the new $\mathscr{P}$, the cost of the overall learning phase be reduced, and the search procedure efficiency be improved. This falls under the scope of another domain of ML called Transfer Learning and developed in Section 2.2.4.2. Before we get into this, we need to understand how the procedure search is performed. The next section explains the basic principles of $\mathscr{P}$ exploration in the case of RL.

1.2.2.2 A Feedback Value to Guide the Search

The second important thing to consider with the ML approach is that the agent search procedure S must be able to distinguish, within $\mathscr{P}$, bad candidate behaviours P from good ones, that would be eligible for the final approximation $\hat{P}$. In other terms, the agent needs a mean to *evaluate* each tested procedure, and tell whether it has been doing well. Evaluation is useful for several reasons. First, it makes it easy not to try bad behaviours again, because S remembers that they have evaluated badly. Second, it makes it possible to

decide when to terminate the search: if a candidate behaviour is evaluated to be sufficiently good, we consider that $\hat{P}$ has been found and there is no need to keep exploring $\mathscr{P}$. In shorter terms, the learning phase is over. Third, evaluation is used as a clue to prioritize the search space $\mathscr{P}$, and browse areas that often yield good evaluations first. This will be developed in Section 2.2. In the case of RL, evaluating the procedures generated by S is a difficult problem, and the field takes a particular approach to it.

In RL, the agent designers build up an additional, special input used as a *feedback*: f. This input is special because it is not supposed to feed the inner procedure P directly like regular inputs, but the procedural search S instead. In a nutshell, i is input to procedure P, whereas f is input for procedure S:



The feedback f is a continuous assessment whether or not the agent is currently doing well. For instance, the user finds the sticky roverbot successful when it faces them, and is close from them, but not too close or else it would be too much "sticky". As such, the feedback signal f is defined with respect to distance and angle to the user by, say:

$$\varphi : \underbrace{-\mathit{distance}}_{\text{stick to user}} \underbrace{+\ln(\mathit{distance})}_{\text{but not too close}} \underbrace{+e^{\cos(\mathit{angle})}}_{\text{and face them}}$$

This yields the feedback profile shown in Figure 1.1-Right. The higher the feedback value, the better the current behaviour evaluation. On this profile, we read that it is desirable for the roverbot to be approximately 1 meter close from the user, because being closer or further yields a lower feedback value. We also read that it is preferable that the bot faces the user direction, because the highest value are attained when the direction is close from $0° \equiv \pm 360°$.

The feedback values make it possible for the search procedure S to progressively construct a mapping between $\mathscr{P}$ and the measure provided by f. In practice, for each tested candidate procedure P, the agent roughly evaluates how much feedback it yields. High feedbacks values are given by the roverbot designers as a *reward* for being successful, while low feedback values are given like *penalties*. In the end, the behaviour earning the most reward (or the less penalty) is chosen as the final approximation $\hat{P}$.

With this simple principle, RL has become a major area of ML and yielded impressive results. State of the art RL automated search agents learn to play human games [Mnih, Kavukcuoglu, Silver, Rusu, *et al.* 2015; Berner *et al.* 2019], trade [Spooner *et al.* 2018] or animate virtual creatures [Lillicrap *et al.* 2015; Gu *et al.* 2016], in a way similar to our imaginary sticky bot.

Box 1.3: ML Design Space.

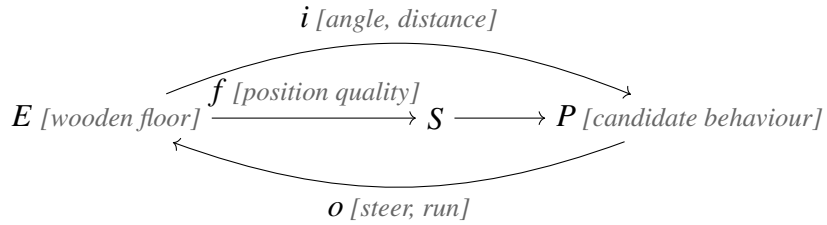
The ML tradeoff is simple. Elusive procedure $P^\star$ are complicated to design. Instead, 3 easier objects are designed: a search space $\mathscr{P}$, a feedback function f, and a search procedure S. The counterpart is that the search within $\mathscr{P}$ is costly, and the resulting $\hat{P}$ that apparently "bridges the AI gap" is not exact. But ML is not trivial. Designing a $\boxed{\mathscr{P}}$ that is both easy to scroll and rich enough to contain acceptable solutions $\hat{P}$ is difficult. A prominent modern solution to this is a tool referred to as Artificial Neural Network (ANN) [McCulloch and Pitts 1943; Schmidhuber 2015]. ANNs are massively parametrized, nonlinear mathematical functions able to approximate any non-ANN function, including $P^\star$, provided they have enough parameters (see Section 2.1.4). Designing $\boxed{f}$ raises other major concerns. For the agent, f is the only clue about our actual *needs* as users. The feedback must be contrasted, so the agent clearly discriminates what not to do. But it must also be unambiguous, or we cannot expect $\hat{P}$ to be satisfying. For instance, the roverbot feedback designed in Figure 1.1 states that we wish the robot to face the user 1 meter close, but does not state whether we wish it to stand in front, or behind them, whether we expect it to remain still if the user does or keep moving around in circle around them. As a result, all these behaviours are possible outcomes of the search. The ML agent designer is responsible to clearly define their goal so that $\hat{P}$ is eventually meaningful. This is the contract embodied by the signature of S: a meaningful $\hat{P}$ is only produced if meaningful f is fed in. Finally, designing $\boxed{S}$ so it makes the best use of available information is still the object of ML science. On the one hand, S must yield good approximations $\hat{P}$ for the result to be satisfying, so it must search $\mathscr{P}$ thoroughly (*correctness* concern). On the other hand, S must perform parsimoniously to alleviate the costs of the "learning phase" (*efficiency* concern). In addition, we expect that S is not only able to solve the task at hand, but also *any* task given a correct f (*genericity* concern). With the same S and a different f, the ideal roverbot is also able to learn how to hide from the user, systematically block their way, or only move around if they are not. A *generic* procedure S, able to find an acceptable $\hat{P}$ in any learning situation, constitutes a dream AI machine able to learn any elusive procedure $P^\star$ that humans cannot write. The perspective of designing a generic learning agent makes ML appealing to both scientists and engineers. In this thesis, we address how signature changes in P transform $\mathscr{P}$, and how changes in f transform S. We also investigate how particular S designs help in efficiently accommodating this phenomenon.

1.2.3 The Signature Change Problem

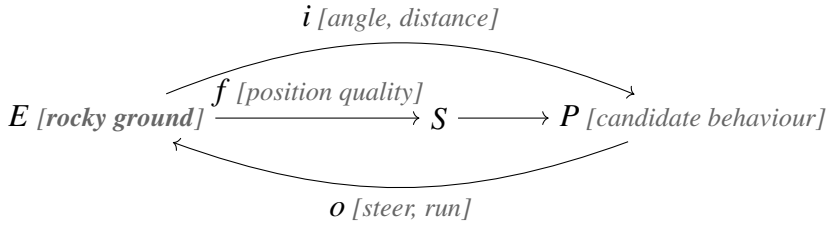
One desired property of ML agents is that they are generic enough to seamlessly tackle various learning situations (see Box 1.3). This is difficult because of the diversity of learning situations. For instance, either the learning context in which S performs its search is fixed, or it varies over time during the course of learning. In the latter case, new challenges must be faced by the agent.

1.2.3.1 Environmental Change

Our sticky roverbot is embedded into a wide, unpredictable environment. As such, unexpected change occasionally occur during the course of learning. For instance, when it starts following the user on a clear, wooden floor, the situation looks like:



Now, when the user goes outside, the situation changes because their yard is rocky. This is an *environmental change*. For later convenience, we refer to this event with the short symbol $(\sim E)$:



At this point, the task has not changed from the user perspective: *"just stick around"*. However, there are chances that the agent be confused, because it has never been trained to face the environmental snags in the yard, and the procedure supposed to fulfill the task has become much more involved. On the wooden floor, sticking around mostly implied locating the user, facing them and running towards them. Now, sticking around also implies locating obstacles, avoiding them, dealing with the crumby floor when the wheels slip, correct the trajectory when drifting with the wind, *etc.*

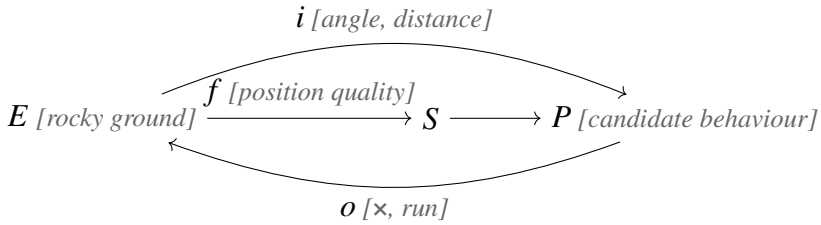
Interestingly, the task has not changed from the agent perspective either: *"pick a procedure $\hat{P}$ that yields good feedback f"*. The problem is that procedures that were good on the wooden floor possibly give terrible results now on the rocky ground. In other terms, the environmental change $(\sim E)$ changes the mapping between $\mathscr{P}$ and f: between the candidate behaviours and their evaluated quality. In this situation, it typically reduces the maximum value that f ever attains, and the adequate procedures $\hat{P}$ becomes much more difficult to find within $\mathscr{P}$ that they were before the change.

Overcoming environmental changes is the subject of ongoing research in ML. Depending on the change (small or big, gradual or sudden), the problem is tackled under various names like Concept Drift [Tsymbal 2004] or Transfer Learning [Taylor and Stone 2009]. Within this scope, the main objective is to make the search procedure S flexible enough so it is aware of the change, and does not have to undergo a costly learning phase

again after the change: only the new aspects of the new environment should be learned again, and previous progress on the wooden floor should not be discarded.

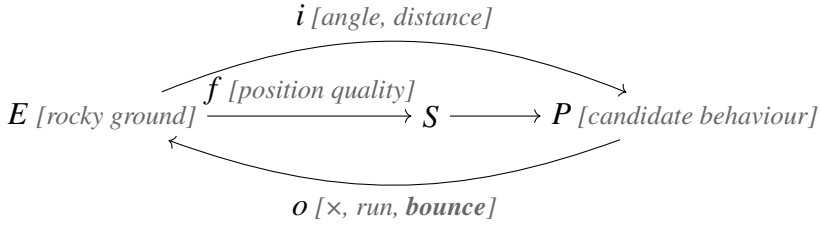
1.2.3.2 Signature Change

Once the roverbot is accustomed to following the user on the rocky ground, another class of unpredictable changes possibly occurs. These are change to the *signature* of the agent procedures P and S. For instance, when the steering bar gets jammed with the dirt, then it cannot control the direction anymore. We refer to this phenomenon as an *output deletion*, and with the short symbol $(-o)$:



This is unfortunate for the roverbot, but the environment and the task at hand remain the same. From the user perspective, the task is still phrased as *"stick around"*, and from the agent perspective: *"keep searching $\mathscr{P}$ for a procedure $\hat{P}$ that yields good feedback f"*. The best it can do from now on is to run frontwards or backwards, along the curve determined by the locked position of the steering bar, so as to keep staying closest from user.

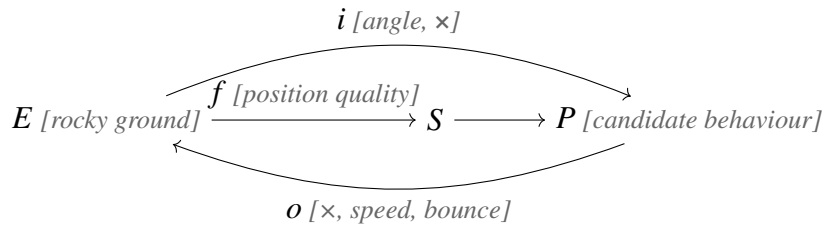
Soon, a new device is plugged into the robot to compensate. Now it is equipped with, say, an bouncy impulse spring. Any time, the bot can use it to jump and end up in a random nearby location, facing a new random direction. We refer to this capability extension as an *output addition* event $(+o)$:



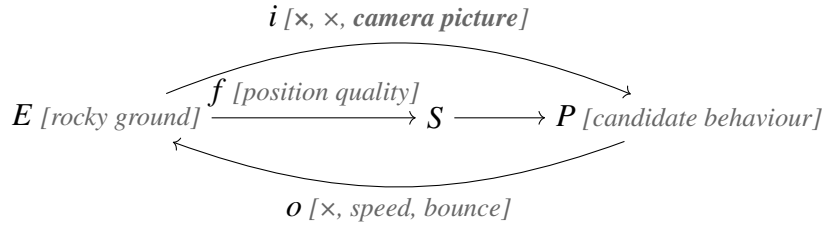
The task at hand still remains the same. But the procedure to fulfill it has to become different again. On the one hand, bouncing around makes it more difficult for the robot to face the user, on the other hand, it makes it easier to escape from blocking rocks.

Later on, the laser supposed to measure the distance to user becomes fickle, and even-

tually fails. This is a case of *input deletion* $(-i)$:

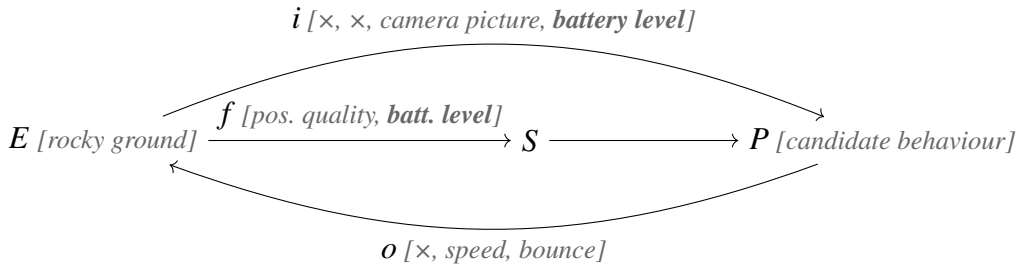


The task remains the same, and the agent still runs towards the user. Only it is more difficult to stop before approaching too close from them. To compensate, the designers decide that P stops feeding only from the angle to the user, which constitutes another input deletion event $(-i)$. Instead, P directly uses the full 1024×1024 pixels picture produced by its front camera on each time step. This has several advantages. For instance, both approximate distance *and* angle to user can be determined from the raw picture. Also, the picture will be useful to spot the obstacles and avoid them before actually hitting them. After this *input addition* event $(+i)$ occurs, the learning situation becomes:



The task remains the same, but fulfilling it now implies that P computes the location of the user and of nearby obstacles from the pixels itself, which represents a much more involved procedure.

Signature changes do not only affect P, they also affect S when the feedback values are updated. In a new upgrade to the roverbot, a new objective is added to the task: watch the bot battery level and go recharge when low. This is made possible with a new feedback value appended, an event called *feedback addition* $(+f)$:

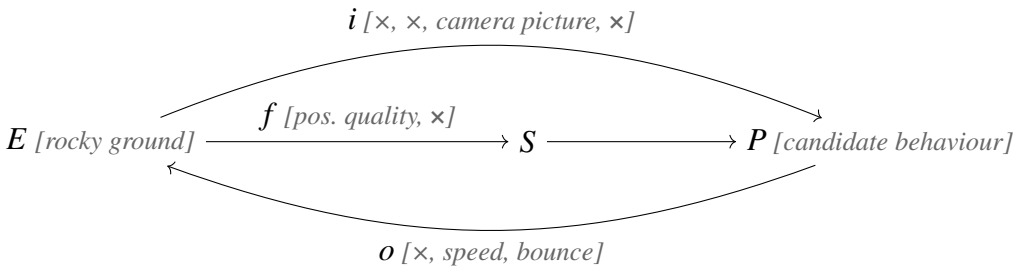


Note that a new battery sensor has also been added $(+i)$.

From now on, not only the position of the roverbot has to be correct, but its battery

level must not become too low, so it has 2 competing objectives to fulfill. This implies that the procedure P be revised again, and the search procedure S has more constraints to satisfy.

Finally, solar panels and powerful accumulators are plugged into the bot, so it does not need to worry about power level anymore. As such, the battery objective can be withdrawn from the agent with a last type of structural signature change event called *feedback deletion* $(-f)$:



Note that the battery sensor has also been removed $(-i)$.

In the end, our imaginary roverbot has undergone numerous modifications since it entered the yard. It is therefore legitimate to expect that its behaviour is much different from what it was on the wooden floor. However, we notice that a few things have not changed during the whole process. First, the environment E has always remained the same since the roverbot entered the yard. Only the bot capabilities evolved. Second, from the user perspective, the task assigned to the roverbot has not changed either: *"stick around"*. This is not exactly true since structural editions made to the feedback signal $(+f,\ -f)$ arguably changed the objectives of the roverbot a little. However, it remains true that the task never changed from the agent perspective: *"search $\mathscr{P}$ for a procedure $\hat{P}$ that yields good feedback values f"*. As such, the user expects that the robot smoothly adjust its behaviour when faced with a signature change.

But signature changes raise a new technical challenge related to the search space $\mathscr{P}$ containing all candidate behaviours. During environmental change $(\sim E)$, the mapping between $\mathscr{P}$ and f was modified, resulting in that procedures that were acceptable before the change possibly ended up yielding lower values. The agent could not tell whether they were still the best ones because the relative values of alternate procedures had possibly raised up, even within the procedures already tried in the past. As such, the agent had to explore $\mathscr{P}$ again, and ML attempts to make this new exploration process the less redundant possible. To this end, ML assumes that not all the mapping has changed, but only a few specific parts to be discovered.

But when the signature of P changes with events like $(+i,\ -i,\ +o,\ -o)$, the situation is much different because *all* behaviours in $\mathscr{P}$ become invalid. Any behaviour in $\mathscr{P}$ was defined with respect to previous signature. For instance, before the roverbot laser failure, all candidate procedures expected 2 inputs values (angle, distance) to process. When the

laser fails, no procedure in $\mathscr{P}$ fulfills the task anymore, as their expected input is incomplete with only (angle). They have become ill-defined and inadequate to the new roverbot capabilities. In shorter terms: any of these changes breaks the *output signature* of S. As a consequence, a new search space needs to be constructed, and the problem of exploring it again with parsimony is, to our knowledge, not tackled by the ML community yet.

The *input signature* of S also changes with events like $(+f,\ -f)$. In this case, the situation is different again because the search space $\mathscr{P}$ is still valid, and so is the mapping between $\mathscr{P}$ and previous values of f. The difference is that the search procedure S needs to be updated so it is now able to process more or less conflicting objectives together. This requires a new, parsimonious exploration of $\mathscr{P}$ again, but with a focus on entirely different challenges, related to multi-objective optimisation and Pareto boundaries [Yaochu Jin and Sendhoff 2008].

In this thesis, we consider that signature change events occurring during the course of learning constitute a new learning situation for automated agents, namely *Protean Learning* (PL). An agent supporting PL enables continuous learning for ML agents undergoing *metamorphosis*, so their capabilities seamlessly evolve over time. At the core of PL, there is the problem of exploring the search space again after each change, with as much parsimony as possible so as not to restart the searching process from scratch. In the second part of the thesis, we focus on $(+i)$ and $(-i)$ in particular.

PL is not only useful for our imaginary sticky roverbot. Like any other ML procedure, the agent algorithm A and its inner search procedure S apply to a broad cast of problems. For instance, consider a long-term automatic trading learner feeding from streaming statistical indicators gathered online [Spooner *et al.* 2018]. Should this agent be erased and restart the learning from scratch whenever a new indicator is created $(+i)$, or when an old indicator is disregarded because not considered relevant anymore by the trading community $(-i)$, then precious computing resources like time, power, data, developers, would regularly be wasted. Instead, the agent must keep on working and improving with the new available information.

1.3 Thesis Outline

In this thesis, we engage into Protean Learning (PL) with several contributions supposed to address the problem of signature change accommodation, with a special focus on input addition and input deletion events $(+i)$ and $(-i)$.

The contributions address two main questions. First, considering that the agent search space is transformed on every signature change: Does a formal definition of PL permit construction of generic accommodation procedures enabling parsimonious reexploration of the search space regardless of the learning context at hand? In Chapter 5, we formalize PL in the general case, then describe an original set of generic accommodation procedures in the restricted case of $(+i)$ and $(-i)$. Second: Do these generic accommo-

dation procedures constitute an improvement over the naive approach of PL restarting the learning from scratch on every change? In Chapters 6 and 7 we address this latter question with an experimental approach aimed to cover the widest range of different learning conditions.

We expect that these contributions be useful to the domain of ML, and for future works in the directions pointed by PL, bringing structural flexibility to automated learning agents.

In the current chapter, we have introduced scientific context to the field of ML, and used a small, imaginary example to introduce all the concepts needed to expose the thesis. In the remaining chapters, we detail the thesis and highlight our contributions to the foundation of PL.

Chapter 2 offers a deeper tour of machine learning, where not only Reinforcement Learning (RL) is addressed, but also Supervised Learning (SL), Unsupervised Learning (UL) and two key transversal learning contexts useful as PL background: Online Learning (OL) and Transfer Learning (TL). Signature changes are not addressed in this contextual chapter, but we stress important common theoretical and technical components of ML, which are used in the rest of the thesis.

Chapter 3 details the signature change problem in depth, along with the need for PL and the scientific challenges involved. Based on this, we formulate our position regarding PL. We defend that PL is a significant, new, and non-trivial domain of ML. In particular, we defend that there exist generic ways to accommodate signature changes in PL regardless of the learning context, the learning method or the task at hand.

Chapter 4 summarizes the state of the art with a range of previous works related to the problem of accommodating changing learning contexts like Continual Learning, Concept Drift, Domain Adaptation *etc.* In particular, we focus on TL to position PL as a particular instance of TL unaddressed *per se* by the ML community yet.

In the second part of this work, we expound our current contributions to PL and our argument to defend the thesis.

Chapter 5 offers a rigorous formalization of the PL situation, useful as a functional model of learning contexts involving signature changes. Then, our focus is restricted to two change events, $(+i)$ and $(-i)$, which constitute the restricted subdomain of Input-PL (IPL). We study the reactions of the explored search space to these signature changes, and show that they are structured by 10 archetypal IPL profiles. In addition, we exhibit a set of natural projections useful to accommodate $(+i)$ and $(-i)$ events regardless of the profile at hand, and suggest that these projections be good candidates to constitute generic IPL techniques.

In Chapter 6, we verify this claim with a controlled synthetic experiment, in an Online SL (OSL) context. We design a testbed involving the most generic type of IPL profile, and compare them to degenerated profiles used as a baseline. Using this testbed, the perfor-

mances of protean agents after $(+i)$ and $(-i)$ events are compared to the performances of a naive agent resuming the learning from scratch on a signature change. We show that the natural projections defined in Chapter 5 provide better results than the naive approach in most situations, and qualify this result depending on the contexts controlled by the experimental parameters.

In Chapter 7, we design and conduct another synthetic experiment to extend the above results to a full-fledged RL context. The testbed we design addresses various IPL profiles and two different learning methods, but the constructed protean agent always use the same natural projections to accommodate $(+i)$ and $(-i)$ events. Once again, we show that their performances outperform the naive agents in most situations, and we qualify the result depending on the various controlled properties of the task at hand.

Chapter 8 eventually wraps up the whole document, with a special highlight on the limits of current conclusions, and the perspectives for future works and future milestones in ML.

Chapter 2

Background: Machine Learning

This chapter introduces the various Machine Learning (ML) approaches, their strengths and limitations, with a special stress on Online Learning situations and Reinforcement Learning, which will be used in subsequent chapters.

Section 2.1 describes a few important common aspects shared among every ML field, because they all address the problem of searching an appropriated computer procedure $\hat{P}$, or "behavioural search". In turn, we expound the maximization problem, the methods — heuristics methods in particular — used to address it, the structure of the associated search space $\mathscr{P}$ and one common technique, namely Artificial Neural Network (ANN), used pervasively throughout ML to implement them.

Section 2.2 describes the distinguishing features of various ML subdomains. Every subdomain specifies its own learning context, in which the behavioural search problem occurs. In Supervised Learning (SL), the user exploits practical samples of the behaviour they look for. In Unsupervised Learning (UL), there are no such samples. In Reinforcement Learning (RL), the behavioural search is coupled with a reactive environment. Some fields of ML are also transversal to others. Online Learning (OL) covers every situation where the progression of behavioural search depends on the progression of external data streams. In Transfer Learning (TL), the behavioural search extends over multiple tasks.

This chapter constitutes general background of the thesis. As such, the particular problem of PL and signature changes, the very object of our work, is not formally addressed until the next chapter.

2.1 Behavioural Search

As explained in chapter 1, the ML approach to AI relies on one cornerstone process: the exploration of a search space $\mathscr{P}$, supposed to contain good approximations $\hat{P}$ of the elusive procedure $P^\star$. Since machines behaviours result from their inner procedures, we refer to this cornerstone process as a *behavioural search*. This section reviews the most widely used tools, methods and approaches to behavioural search.

2.1.1 The Maximization Problem

The behavioural search required by ML is a particular case of a longstanding problem occurring throughout sciences in general, and computer science in particular: the *maximization problem*. In this problem, there is an *objective* function, our "feedback" function f, mapping each element of the search space to a numerical value:

$$f: \begin{cases} \mathscr{P} \to \mathbb{R} \\ P \mapsto f(P) \end{cases} \tag{2.1}$$

The problem is to find which elements in the search space yield the maximum possible value. In other terms, the desired elements P^* are the solutions of the equation:

$$f(P^*) = \max_{P \in \mathscr{P}}(f(P)) \tag{2.2}$$

Note that it is not strictly necessary for f to output numerical values in $\mathbb{R}$. The only requirement is that the feedback values be at least partially ordered, so it makes sense to look for elements P^* for which no other candidate P yields strictly better feedback. Considering this, the best elements P^* are more generally specified by:

$$\forall P \in \mathscr{P},\ f(P) \leqslant f(P^*) \tag{2.3}$$

Also note that P^* is not necessarily unique within $\mathscr{P}$, but there is always at least one.

Arguably any problem, and any computer science problem in particular, eventually reduces to a maximization problem. To reduce it, consider that $\mathscr{P}$ virtually represents any possible solutions set, and that f always measures, at the very least, whether the user is satisfied with the solution (say, $f(P^*) = 1$) or not (say, $f(P) = 0$). In the particular ML situation, f measures the quality of approximation $\hat{P}$ with respect to the targeted "elusive procedure" $P^\star$. As a consequence, no matter the considered ML technique, the resulting behaviour $\hat{P}$, used as a proxy to $P^\star$, is, at best, a solution to equation (2.3). At best $\hat{P} = P^*$.

The maximization problem is hard to solve in general[1]. This means that, given $\mathscr{P}$ and f, it is not always possible to find the maximal candidates P^*. Instead, ML users fall back on using the best candidates $\hat{P}$ found so far.

In summary, the quality of $\hat{P}$ depends on three major factors:

- $\mathscr{P}$: The search space defined by the user. No good behaviour $\hat{P}$ can be found if $\mathscr{P}$ does not contain any in the first place. The design of $\mathscr{P}$ is discussed in Section 2.1.3.
- f: The feedback function defined by the user. No good $\hat{P}$ can be found if f does not accurately represents the candidates quality (see Box 1.3).
- $(A,\ S)$: The agent search procedure defined by the user. No good $\hat{P}$ can be found if the search procedure fails in exploring $\mathscr{P}$ and misses important candidates. The next section describes search procedures commonly used in ML.

[1] Otherwise it would be easy to solve any problem.

2.1.2 Heuristics

Analysis The most rigorous method for behavioural search is the mathematical analysis of the objective function f. For example, if the agent behaviour is only parametrized by one scalar value P, and given the concave objective function $f\colon P \mapsto e^{P-P^2}$, we can calculate that the only solution to $f'(P) = 0$ is $\frac{1}{2}$, and that $f''\left(\frac{1}{2}\right) = -2e^{\frac{1}{4}}$ is negative, so f admits exactly 1 maximal candidate $\hat{P} = P^* = \frac{1}{2}$.

When a $\hat{P}$ is found with this method, it is guaranteed to solve eq. (2.3). In the above case, $\hat{P}$ is strictly a best candidate for approximation of $P^\star$, unless either f or $\mathscr{P}$ were ill-designed to address the problem at hand. However, in most real-world ML problems, the candidates P are not simple numbers but computer programs represented with large vectors or complex objects. And feedback functions f are not concave but discontinuous mappings including various complicated forms of non-linearity.

As a consequence, the analytical approach suffers the major issue that it requires deep theoretical groundwork, and that the majority of prominent ML problems are intractable in practice. Reasons for this include undecidability, lack of mathematical knowledge, combinatory explosion of the number of possible cases to study, or lack of computational resource. As a consequence, the exact solutions P^* to eq. (2.3) cannot be found. One possible workaround is to emit simplifying hypotheses, but this degrades in turn the quality of $\mathscr{P}$ and f, thus the final behaviour $\hat{P}$.

Inexact Methods One alternate approach to behavioural search, broadly employed throughout ML, is to rely on inexact methods. Inexact methods are also called *heuristics* because their results is not guaranteed to be optimal, but they practically reach sufficient approximations[2]. In heuristics, no complete analysis of $\mathscr{P}$ is done. Instead, a large number of particular candidates P is drawn from $\mathscr{P}$, and evaluated with $f(P)$ so as to determine their quality. When a satisfactory candidate is eventually found, it constitutes the final procedure $\hat{P}$. Compared to analytic approaches, heuristics suffer the major pitfall that $\mathscr{P}$ is not entirely scanned. As a consequence, many candidate procedures are never actually tried, and there is no guarantee that any *true* solution to eq. (2.3) is eventually found. These approaches must therefore be ruled out whenever an exact result is absolutely required. On the other hand, heuristics make it always possible to explore $\mathscr{P}$ at least a little. And the pragmatic results $\hat{P}$ yielded by these techniques turn out to be useful in practice[3].

[2] In this respect, they are more general than, and not to be confused with the restricted sets of "admissible and non-admissible heuristics" used in the particular domain planning and pathfinding.

[3] Heuristics are typically inspired from natural processes, because some natural processes are easily interpreted as behavioural searches. One famous instance is the *adaptation* capabilities of species. Evolution shapes species in a way that their reactions to environment continuously update [Darwin 1859]. The associated biological process, a.k.a. natural selection, has inspired procedural searches from the *local heuristics* category described hereafter. *Ontogenesis* is another example of natural behavioural search, since individuals progressively learn to behave in an environment they initially know nothing about. This process is much more complex as it also involves inherited priors and observational learning [Bandura and Walters 1963; Zentall 2012]. This falls out of scope for this thesis as we focus on single-agent search.

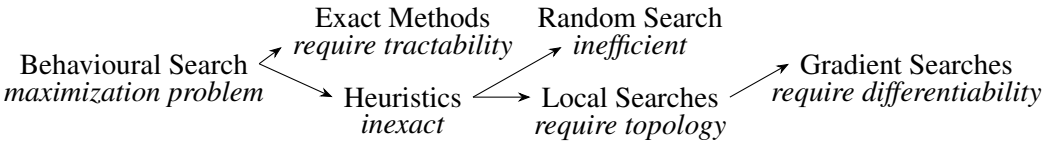


Figure 2.1: Various categories of behavioural searches.

Heuristic searches boil down to the repetition of many elementary operations called heuristic *steps*: "pick P from $\mathscr{P}$ then evaluate $f(P)$". Given a fixed step cost in terms of computational resources, the more steps performed before a satisfying $\hat{P}$ is found, the more expensive the approach. The various heuristic methods described hereafter (see Figure 2.1) share one common objective: reducing the number of steps required before a satisfying $\hat{P}$ is eventually found. In particular, they differ on the way the next candidate P is chosen on each step.

In its most naive form, heuristic behavioural search is made at random: the next candidate P is always randomly drawn from $\mathscr{P}$. This approach only requires that a random distribution be defined over $\mathscr{P}$, but is likely to be inefficient in practice, because many unlucky samplings are done before an interesting $\hat{P}$ is eventually found, especially when the interesting candidates $\hat{P}$ are rare within $\mathscr{P}$ or underrepresented in the chosen distribution.

Local Heuristics Instead, one major improvement over random search is to *guide* the search towards promising regions of the search space. This requires that $\mathscr{P}$ be supplied with a *topology* structure, and is a cornerstone aspect of the *signature change* problem addressed in the next chapters. The topology provides a notion of *neighbourhood*, so it makes sense to say that some candidate procedures P are more or less "connected" to each other. In addition to topology, the search space is equipped further with a notion of *distance* or *dissimilarity*, so it also makes sense to say that candidate procedures are more or less "close" or "similar" from each other, and speak of "regions" of $\mathscr{P}$ like an actually spatialized set.

Once this structure is defined, the idea is to hypothesize that good quality procedures neighbour each other within $\mathscr{P}$. In other terms, f is assumed to vary *smoothly* over $\mathscr{P}$ so that, whenever a good candidate is found, there is a fair chance that better candidates be located around it. With this set up, then the next tested P is preferably drawn from the neighbourhood of the best candidates found so far. This reduces the number of steps eventually needed to find a satisfying $\hat{P}$ within $\mathscr{P}$.

In this situation, the profile of the objective function f is commonly nicknamed the "landscape" of f, with high-valued regions pictured as hills, and low-variability regions as plateaus. The search procedure is also nicknamed "hill-climbing" procedure. Note that the spatial structure of the search space must be defined in a way that the smoothness hypothesis holds, otherwise the above technique does not yield better results than plain random search. We refer to this particular category of heuristics as *local searches*.

Local searches encompass many popular algorithms. In "evolutionary algorithms" for instance, the less promising candidates P are successively filtered out from many generations of candidates, while new generations are created from local modifications of previous ones, according to principles of natural selection [Sastry *et al.* 2014]. They have been successfully applied to Reinforcement Learning in [Moriarty *et al.* 1999], to the concept drift problem in [Rohlfshagen *et al.* 2009], and there exist modular approaches to it [Potter 1997]. In "simulated annealing", the new candidates are first drawn from very distant regions, then the sampling regions are gradually narrowed around the most promising candidates as the "temperature" parameter cools down, until only high quality candidates remain [S. Kirkpatrick *et al.* 1983]. "Ant colonies" algorithms use the same locality principles, in a way inspired from emergent behaviour of eusocial species [Maniezzo 1992]. See [Osman and Laporte 1996] for an insight into the diversity of these local approaches.

Exploration and Exploitation Local heuristics share a common challenge, well-known as the *exploration vs. exploitation* dilemma [Gupta *et al.* 2006]. As a matter of fact, the landscape of f typically exhibits *local optima*: non-optimal candidates that dominate their regions, but are less good than the *global optima* which verify eq. (2.3). This is misleading to the search procedures, since they are attracted towards any promising region, but they have no way to tell whether the optima lying within the region are local or global.

The exploration / exploitation dilemma is an important balance to strike between two contradictory search activities. On the one hand, promising regions of $\mathscr{P}$ need to be thoroughly sampled to find the best candidates among them. This requires many *exploitation* steps. On the other hand, the smoothness hypothesis never guarantees that a best candidate in a promising region best over all $\mathscr{P}$. The only way to address this is to examine other, further regions, and check whether they yield better candidates. This requires in turn many *exploration* steps. To summarize: if *exploration* is missing, then important parts of $\mathscr{P}$ are wasted. But if *exploitation* is missing, then no excellent candidates are found within the promising regions. However, performing the two is computationally expensive. Satisfying this tradeoff, depending on the problem at hand, is a major concern of local searches, and also the object of ongoing research across the various fields of ML, including RL.

Gradient Searches Local searches can be improved further under some particular circumstances. In the above methods, the region surrounding each promising candidate is sampled arbitrarily to find some promising neighbourhood to explore next. This "local exploration" costs multiple steps. But if $\mathscr{P}$ is a *continuous* set and f a *differentiable* function, then there is a vector directly pointing towards the next most interesting neighbourhood. This vector is built from the derivative value $\frac{\mathrm{d}f}{\mathrm{d}P}(P)$. With this value calculated, the search algorithm just needs to follow it and saves many local exploration steps.

The methods relying on this particular clue, like gradient ascent or adam [Kingma and Ba 2015] are called *gradient searches*. Gradient searches imply that f can be at least approximately differentiated at the current candidate location P, but they dramatically

increase the search performance, especially in smooth, high-dimensional search spaces. They are especially useful with the ANN tools described in Section 2.1.4.

Parametrization Most heuristic methods need to be *parametrized*. In adam, for instance, a parameter called "learning rate" specifies how far to follow the gradient direction on each step. In simulated annealing, the "cooling" parameter specifies how fast the temperature decreases. Heuristics are typically sensitive to these parameters, in the sense that the quality of the resulting $\hat{P}$ strongly depends on them. But the parameters are difficult to tune to correctly match the landscape at hand. No predefined methodology precisely determines which parameter values to use, so they need to be groped, which makes heuristics difficult to apply on new problems. In spite of this limitation, local searches yield consistent and useful results in practice, and are widely used in ML.

When the signature of the targeted procedure $P^\star$ changes, the whole search space $\mathscr{P}$ needs to change, and whichever search method used by the learning agent needs to accommodate the event because its target has transformed. This is the particular situation we address in this thesis. Chapter 5 describes the general connections between search landscapes when $(+i)$ and $(-i)$ events occur, and projections applicable to any kind of behavioural search. Chapters 6 and 7 address accommodation of these change events by three different local searches: adam search (with RNN learners), Q-Learning and Actor-Critic (in a Reinforcement context).

2.1.3 The Search Space

There exists various ways to define the search space $\mathscr{P}$. They have in common that $\mathscr{P}$ must be both wide, so it has chances to contain good final candidates $\hat{P}$, and easy to explore, so these candidates have good chances to be found.

Naively, the largest search space is the space of all possible computer programs. This space is huge, and it certainly contains the procedures desired by the user, but it is hard to explore for several reasons. With exact search methods, this space is intractable because of combinatory explosion and the many undecidable properties of computer programs. With local heuristics, changing one small directive in a computer program typically makes it behave a totally different way, so it is hard to equip this space with a topology and a distance measure that makes the feedback function f vary smoothly over it. As a consequence, the only unaffected automated search procedures over this space belong to the random heuristics category, and are extremely inefficient because the subset of useful programs is extremely sparse within the set of all possible programs.

In most cases, the search space must therefore be reduced to be more easily explored. This implies that only a subset of all possible procedures be selected, a.k.a. $\mathscr{P}$, and others be left aside. Selecting this subset is non trivial, and there are two major approaches to it.

In *mechanistic* approaches, an insight into the inner mechanics of the unknown procedure $P^\star$ is used, so $\mathscr{P}$ only contains procedures whose internals are likely to resemble the internals of $P^\star$. For instance, when looking for a procedure that correctly predicts the incidence of a pandemic disease given the time $i = t$ elapsed since primary infection, the set of logistic functions $\mathscr{P} = \{P\colon i \mapsto \frac{C}{C+(1-C)\,e^{-ri}}\}$ is an insightful space to search [Prentice and Pyke 1979]. If a good $\hat{P}$ is found in this space, then the corresponding value of C corresponds to the initial infected pool, and r measures the disease propagation rate. Mechanistic approaches are compelling because of the strong semantics they provide, but they also require strong prior knowledge about $P^\star$, which is uncommon to have in ML.

In *phenomenological* approaches, on the other hand, $\mathscr{P}$ is constructed so that it loses as few information as possible compared to the larger set of all possible procedures. In an ideal phenomenological $\mathscr{P}$, any computer procedure (including $P^\star$), can be closely approached from at least one candidate in $\mathscr{P}$. This guarantees that good candidates $\hat{P}$ exist, even though $\mathscr{P}$ is easier to search. For instance, polynomial functions can approach any function arbitrarily closely, provided their degree D is high enough and their coefficients are correctly set (see next section). In addition, the space of polynomial functions is easy to explore because it is it in bijection with the space of coefficients, so $\mathscr{P} \longleftrightarrow \mathbb{R}^{Df}$ with $Df = D + 1$ degrees of freedom.

In this situation, $\mathscr{P}$ is *de facto* equipped with a strong topology that enables local heuristics, and even gradient searches with the most common differentiable feedback functions f. Phenomenological approaches are the more generic, because they do not rely on insights into $P^\star$. As such, they are by far the most used in ML. In the remainder of this thesis, we only address phenomenological techniques because they are the most common, but see Box 2.1 for a more detailed comparison with mechanistic approaches.

2.1.4 Artificial Neural Networks

Arguably, the overall success of modern ML stems from one particular class of tools, dedicated to phenomenological approximation, whose tremendous performances are responsible for the most remarkable achievements in this field. These tools are called Artificial Neural Networks (ANNs) [Haykin 1999].

Like polynomials, ANNs approximate any function provided they have enough degrees of freedom Df and their coefficients are correctly set. Like polynomials, they are parametrized by one vector of numbers so the associated search space is typically $\mathscr{P} \longleftrightarrow \mathbb{R}^{Df}$, which enables local heuristics. Like polynomials, they are typically differentiable, which enables efficient gradient searches. In addition, they represent a broader class of functional forms than polynomials, they are more flexible and more easily customized to fit particular uses. For instance, they are not restricted to represent deterministic procedures, as they can also be stochastic [Liao and Mao 1996] or recurrent [Mandic and Chambers 2001; Lipton 2015].

The formulae of ANNs are typically deeper and more intricate than the formulae for

Box 2.1: Mechanistic *vs.* Phenomenological Approaches

There are two attitudes regarding definition of $\mathscr{P}$. In *mechanistic* approaches, $\mathscr{P}_{mc}$ only contains meaningful procedures, understandable by humans. In *phenomenological* approaches, $\mathscr{P}_{ph}$ contains many procedures with various behaviours, not representing anything. $\mathscr{P}_{ph}$ advantage is to almost certainly contains a good $\hat{P}$. But $\hat{P}$ may produce aberrant results given unusual input. Conversely, $\mathscr{P}_{mc}$ guarantees that $\hat{P}$ is consistent with $P^\star$. $\mathscr{P}_{mc}$ has a scientific advantage because it carries explicit hypotheses about $P^\star$, and *conclusions* are drawn from $\hat{P}$: If $\hat{P}$ is bad, hypotheses are safely dismissed; If $\hat{P}$ is good, hypotheses are reinforced. As a downside, $\mathscr{P}_{mc}$ requires clever insights into $P^\star$, which require difficult theoretical groundwork, about the "elusive procedures". This makes $\mathscr{P}_{ph}$ appealing. The model they use (polynomials, ANNs) do not capture essential *mechanics* of $P^\star$, only the observed *phenomenon*, so they seamlessly approach any $P^\star$. $\mathscr{P}_{ph}$ is easily misused as a *replacement* to $\mathscr{P}_{mc}$, but the two approaches are disjoint and complementary. As a phenomenological approach to AI, ML outcomes must be treated with care and not over-interpreted as models of the human mind. Nevertheless, it is a matter of fact that satisfying mechanistic models of $P^\star$, like human speech recognition, are lacking. And the rise of ML proved "weak" $\mathscr{P}_{ph}$ predictions to be useful in practice. This is because ML searching procedures are tuned with care, and much thought is actually devoted to alleviating their phenomenological limitations. For instance, the ANNs used in picture recognition are not vanilla ANNs but CNNs, inspired from visual cortex, because their formula is adapted to image processing (detecting contours, overlapping areas) [Krizhevsky *et al.* 2012]. Unveiling CNNs constitutes an involved groundwork and diminishes genericity, but increases efficiency and understandability. In other terms, ML reveals, with flexible ANNs tools, that there is a continuum between phenomenological and mechanistic approaches. On the phenomenological end, agents are flexible and learn without much human input, but it is unclear how they work and they are subject to aberrations. On the mechanistic end, agents require theoretical groundwork and are hand-crafted with reduced autonomy, but yield sounder results and humans understand their functioning. The question whether it is possible for an automated learning agent to yield both autonomous and intelligible results is still open. And generic learning models that strike, like human learners, the right balance along this gradient are the object of ongoing research. This thesis addresses signature changes in any attitude, although the two conducted experiments are held within the scope of phenomenological ML.

polynomials, so the computation of the derivatives necessary for gradient searches constitutes a more involved process. This differentiation process relies on thorough applications of the differentiation chaining rule, which, in this context, is commonly called "backpropagation" instead [Haykin 1999].

The principle of ANNs is simple. Here, we introduce them as an extension to polynomial approximators. The formula for polynomials is progressively built by the summation

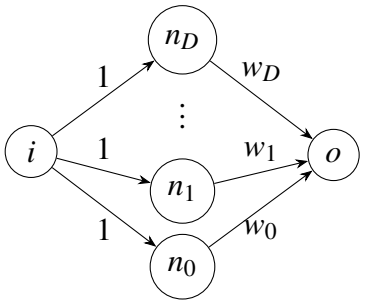
of successive "monoms" of the form:

$$m_d(i) = w_d\, i^d \tag{2.4}$$

With $d \in \mathbb{N}$ the monom degree, and $w_d \in \mathbb{R}$ the associated multiplying *coefficient*, also called "weight" . The final formula for a polynomial of degree D is:

$$o = P(i) = \sum_{d=0}^{D} m_d(i) = \sum_{d=0}^{D} w_d\, i^d = w_0 + w_1\, i + \cdots + w_D\, i^D \tag{2.5}$$

The formula is entirely specified with a vector of coefficients $w \in \mathbb{R}^{D+1}$. In the context of ANNs, it is common to use labeled, directed graphs to represent such formulae:

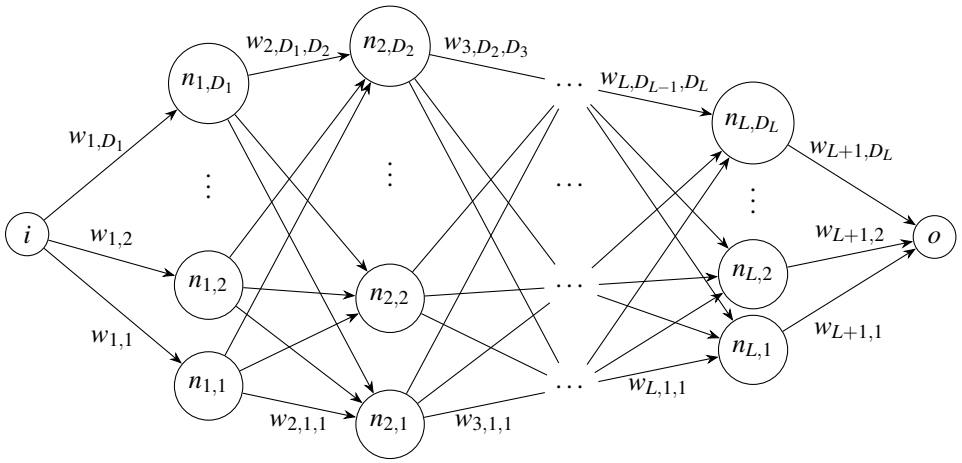


(2.6)

In this representation, node labels represent functions, and arrow labels represent weights. The value of each arrow tip is the value of its source node multiplied by its label, and the value of each node (or "neuron") is the sum of incoming arrows tips transformed by the node label. Here, i and o correspond to identity functions, in particular, *input* and *output* of the procedure, while each n_d is the power function:

$$n_d : x \mapsto x^d \tag{2.7}$$

ANNs generalize the polynomial formula to any formula described by directed graphs. In particular, the typical structure of traditional ANNs, loosely inspired from cortical neuron organization, is a multilayered formula, where each layer is only connected to the two adjacent ones. With L layers, the formula looks like:



(2.8)

Note that the number of nodes per layer $(D_1, D_2, \ldots, D_L)$ typically varies within the same network. However, the function associated with every intermediate node $n_{l,d}$ is traditionally the same: one non-linear function called an *activation function*, like $x \mapsto |x|$ or $x \mapsto \tanh(x)$. Even with this simple structure, many parameters are needed to specify the procedure, which makes the associated search space $\mathscr{P}$ wide in terms of dimensionality. And the many possible computation paths among the network makes it possible for the formula to exhibit various different behaviours, some of which are likely to approximate the desired procedure $P^\star$. In fact, *Universal Approximation Theorems* guarantee, for each type of ANN, that a large category of functions can be approximated this way [Haykin 1999; Mhaskar and Poggio 2016; Zhou 2020].

Possible modifications to this simple structure extend ANN to broader classes of procedures. For instance, additional "bias" nodes, with no incoming arrows, can be added to each layer to shift all inner values by a constant amount. Alternately, the activation function represents the last transformation applied to produce a node output, and can be adjusted with fine grain on each node to fit the problem at hand. On the input side of the nodes, all incoming valued are traditionally integrated as a sum, but this can be extended to any aggregate function like mean, max, *etc.* This is useful to construct one particular class of ANNs called Convolutional Neural Networks (CNNs), well-suited to process autocorrelated input data like pictures [Lecun and Bengio 1995; Zhang *et al.* 2017; Dhillon and Verma 2020]. The network structure itself can become arbitrarily complex, and even be part of the search space itself in approaches called "NeuroEvolution of Augmenting Topologies" (NEAT) [Stanley and Miikkulainen 2002]. Note that a search space featuring various network structures cannot be explored with gradient-based approaches due to the discrete nature of directed graphs, so NEAT exploration needs to partly fall back on more generic local searches like genetic algorithms. Finally, it is possible to correctly define *cycling* network structures, provided at least one arrow on each cycle is associated with a positive *delay*, meaning that several passes are needed to propagate values throughout the network. Cycling networks are called Recurrent Neural Networks (RNNs) [Mandic and Chambers 2001; Lipton 2015] and constitute non-Markovian procedures, useful to approximate processes exhibiting a memory like natural language [Elman 1990; Cho *et al.* 2014], or even NEAT search procedures themselves [Zoph and Le 2016].

RNNs in particular are typically well-suited to Reinforcement Learning, but suffer one major concern known as the *exploding/vanishing gradient* problem. In theory, their cycling structure permits that any time-dependent procedure be approached, no matter the time scale. But in practice, gradient searches suffer from limitations of the backpropagation process. When computing the derivative of the network by applying the chaining rule up to far back in time, algorithms commonly loose track of the derivative value, either because its magnitude weakens, or because it explodes due to artefactual cumulation of rounding errors [Pascanu *et al.* 2013]. Various RNN structures have been offered to alleviate this. For instance, Long Short-Term Memory networks (LSTM) have been popular for a couple of decades [Hochreiter and Schmidhuber 1997; Bakker 2001].

And more recently, a new architecture called Gated Recurrent Unit (GRU) has proved to achieve the same performances with a smaller coefficients space [Cho *et al.* 2014].

Arguably, the possible extensions of ANNs make the set of possible networks almost as wide as the set of possible computer programs, since directed graphs like the one shown in equation (2.8) are representations of arbitrary computer calculations in general. This seems to contradict with the need for reducing the search space. In fact, the strength of ANNs is threefold. First, the network structure is flexible enough to fit various classes of problems, and it is possible for the ML user to precisely customize their own reduction of the search space with a dedicated ANN. Second, universal approximation theorems guarantee that certain simple classes of ANNs structures, like the multilayered structures, are sufficient to closely approximate arbitrary functions. As a consequence, the structure may be fixed during the search without much loss in approximation capabilities, and there is no need to explore the space of all structures anymore. Third, once the structure is fixed, the search space is reduced to $\mathscr{P} \longleftrightarrow \mathbb{R}^D f$, which makes it possible to use efficient local heuristics like gradient searches, whereas this is not possible when directly searching the space of possible computer programs. Even when the structure varies during the search, with NEAT-like approaches, gradient searches are performed between structural edition steps, which strongly speeds up the algorithms.

In the end, ANNs have overwhelmingly proved to be a powerful tool for phenomenological behavioural search, and arguably gave ML the boost that ranked it among the most prominent approaches to modern AI. Note however that they leave mechanistic approaches aside. The strong differences between *mechanistic* and *phenomenological* approaches to procedure approximation constitute an important epistemological insight into the balance between the power and limits of ML (see Box 2.1).

This section has described the optimization problem, the various approaches to it (heuristics in particular), phenomenological and mechanistic attitudes to the design of the behavioural search space $\mathscr{P}$ and one common tool, namely ANN, performing the search with phenomenological approaches. These principles are common across ML.

On the other hand, ML is diverse because it addresses various different learning contexts, and subdivides into several subdomains. In particular, the problem of signature changes we are interested in occurs in some domains of ML, but not in others. The next section summarizes these various subdomains, and emphasize their relations to PL.

2.2 Learning Contexts

ML is commonly considered to branch into three major subdomains: Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). This depends on the context of the elusive procedure $P^\star$ being approximated. In other terms, while the tools and methods described in previous section differed by the way $\mathscr{P}$ was defined and

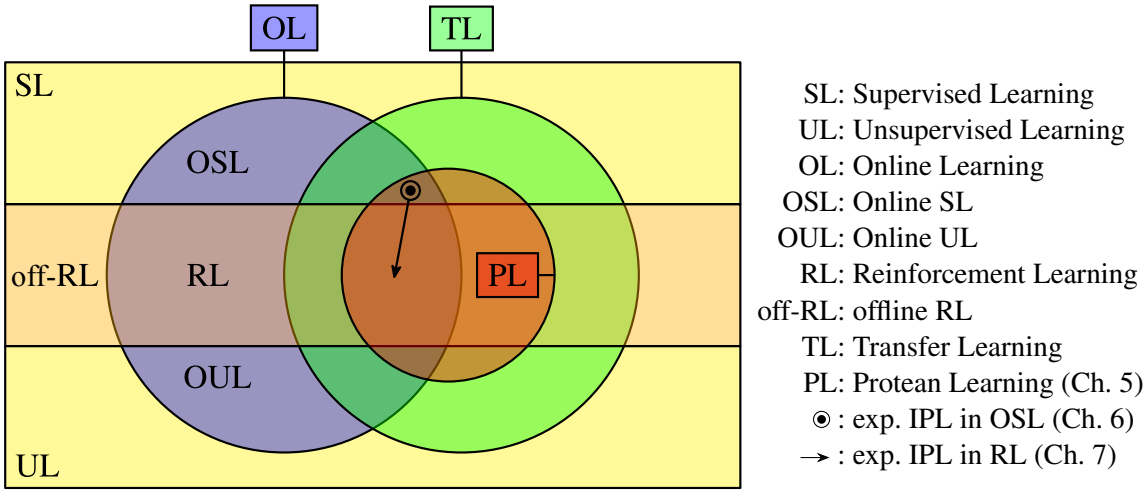


Figure 2.2: Informal map of various ML situations under the form of a Euler Diagram. The red area represents the PL situation defended in this thesis. The dot and arrow represent our experimental contributions to PL, in particular *Input-PL* (IPL) with the experiments exposed in subsequent chapters.

searched, the ML subdomains described hereafter differ with respect to user expectations about the resulting $\hat{P}$. They are illustrated in Figure 2.2 under the form of a Euler diagram.

2.2.1 Supervised Learning

SL is the most straightforward application of ML to the AI problem of "elusive procedures" $P^\star$ described in Section 1.1.

2.2.1.1 The SL problem

In SL, $P^\star$ is a one-shot procedure, feeding from input data i and producing output data o once as a result:

$$i \to P^\star \to o$$

For instance, one common "elusive procedure" is the task of recognizing an object in a picture. In this task, i constitutes a raw picture, say 256×256 pixels with 256 levels rgb channels each, and o is a symbol, or "label", taken from a predefined set of picture categories, say $\{\mathsf{kitten},\ \mathsf{plane},\ \mathsf{banana},\ \mathsf{unknown}\}$. In this regard, the signature of $P^\star$ is well-known from computer scientists. But the procedure $P^\star$ that would correctly fulfill the task cannot be written explicitly out. In a nutshell: humans easily label pictures at first sight, but it is unclear how to process the corresponding pixels automatically so the computation always ends up on the right symbol.

To understand SL approach, it is convenient to represent $P^\star$ as a plain mathematical function, *i.e.* $P^\star\colon i \mapsto o$. This suggests that $P^\star$, like mathematical functions, is *deterministic*. This is not the case in general. For instance, in the picture labelling example, there are

always pictures that humans have trouble classifying as either a kitten or a banana, not because the two objects look alike, but because the image is poor quality, or is an illusion, or clearly features both in a way that the unknown label also seems unsuitable. Depending on their mood, humans pick one category or the other in unpredictable ways.

However, no matter the nature of $P^\star$ (*e.g.*, a function, a random process), SL attempts to approximate it with another procedure $\hat{P}$ whose nature possibly differ. For instance, a non-deterministic process is possibly approximated with a stochastic process, or a random process with a deterministic function. In the end, the goal of SL is only to find an acceptable approximation $\hat{P}$, so we stick to this simple functional representation for the sake of simplicity:

$$o = P(i) \tag{2.9}$$

There is something very specific to the SL situation: in SL, although the agent designers cannot write $P^\star$, they happen to know a large set of examples realisations of $P^\star$, each of the form $(i,\ o^\star)$ with $o^\star = P^\star(i)$. This set is called a *training set*, or $\mathscr{T}$. In the object recognition example, $\mathscr{T}$ constitutes, say, a database with a few thousand pictures, each of which is annotated by a human with a "correct" symbol. The idea of SL is to use this prior data as a hint to the automated agent so it finds a good approximation of $P^\star$, hence the naming "Supervised". To summarize, in SL, the learning agent feeds from the training set as an input, and outputs the approximation $\hat{P}$ as the result of its search:

$$\mathscr{T} \to A \to \hat{P}$$

$\hat{P}$ has the same signature as $P^\star$, so the user can use it to process new images that have not been labelled yet, and expect that $\hat{P}$ correctly discriminates kitten pictures from plane pictures, *etc.* Note that $\hat{P}$ is approximate, so it does make mistakes. And these mistakes cannot be automatically detected before a human eventually figures them out. The only advantage is that $\hat{P}$ is a rigorously predefined computer procedure, so it can label thousands new images within seconds, whereas humans are inefficient to perform the same task again. In a nutshell, SL trades precision for resources. This makes it ill-adapted for use in contexts where absolute correctness matters, like medical analysis, satellite control or governance decisions, but it remains useful in many applications like classifying intricate data [Kotsiantis 2007], performing multivariate regression [Mitchell 1997; Chiplunkar *et al.* 2017], or reproducing artistic patterns [Elgammal *et al.* 2018].

2.2.1.2 The SL Approach

Consistently with ML principles, the SL agent explores a wide set of candidate procedures $P \in \mathscr{P}$, typically an ANN, to find $\hat{P}$. Like the roverbot in section 1.2, it performs inner calls to a search procedure S, typically a gradient-based heuristic, and tests numerous candidate procedures P outputted by S. The search is guided by feedback procedure, or "objective function" $f_{\mathscr{T}} : \mathscr{P} \to \mathbb{R}$.

One noticeable difference is that, unlike RL feedback values, SL feedback values are not outputted by some environment external to the agent. In SL, values of f are directly computed with the help of the training set. For instance, to evaluate a candidate procedure P, f applies P to every input stored in the training set, and compares the resulting $o = P(i)$ to the expected results $o^\star = P^\star(i)$. The more all o values resemble the corresponding $o^\star$ values, the higher the feedback value.

There are various ways to measure "similarity" between the values. When P outputs numbers, for instance, one common comparison is the Mean-Squared Error (MSE) distance between every "predictions" o and the actual training data $o^\star$:

$$f_{\mathscr{T}}(P) = -\frac{1}{|\mathscr{T}|} \sum_{(i,\, o^\star)\in\mathscr{T}} (o^\star - P(i))^2 \tag{2.10}$$

Where $|x|$ is the number of elements in x.

In the picture classification example, another measure of distance has to be used to compare the symbols outputted by P and the symbols outputted by $P^\star$, because they are not numbers. For instance, a function indicating their equality:

$$f_{\mathscr{T}}(P) = -\frac{1}{|\mathscr{T}|} \sum_{(i,\, o^\star)\in\mathscr{T}} \mathbb{1}_{o^\star = P(i)} \tag{2.11}$$

Where:

$$\mathbb{1}_x = \begin{cases} 1 & \text{if } x \text{ is true} \\ 0 & \text{if } x \text{ is false} \end{cases} \tag{2.12}$$

The higher the value of f, the better the currently tested procedure P.

In the end, SL eventually boils down to the problem of finding one $\hat{P}$ in $\mathscr{P}$ that maximizes f, which is an instance of the maximization problem defined in Section 2.1.1. The procedure $\hat{P}$ eventually outputted by the agent is a procedure that yielded best feedback when tested. High feedback values mean that the outputs, or "predictions" $\hat{o} = \hat{P}(i)$ closely resemble the training outputs $o^\star = P^\star(i)$. At this point, SL concludes that $\hat{P}$ is a fair surrogate to $P^\star$. This principle is also known as a statistical *regression*. There exists very sophisticated approaches to phenomenological regression, like multivariate adaptive regression splines [Friedman 1991], ANN approximators, or Gaussian processes [Chiplunkar *et al.* 2017], and "SL" is arguably a fancy name for a regression which happens to use sophisticated phenomenological approaches.

2.2.1.3 The Limits of SL

The strength of phenomenological approaches like SL is that they are always flexible enough to fit the training set. This is the reason for SL success, but also for its most famous pitfall known as *overfitting*.

Designing a wide and flexible $\mathscr{P}$ almost guarantees that it contains good approxima-

tions of $P^\star$. There is also a guarantee that $\mathscr{P}$ almost certainly contains good final candidates $\hat{P}$ with high feedback values $f_{\mathscr{T}}(\hat{P})$. But these two guarantees are not the same: the latter is weaker than the former. In other terms, it is easier to find candidates that yield good feedback $f_{\mathscr{T}}(\hat{P})$, because $\hat{P}$ only needs to closely fit the training points $(i,\ o^\star)$, than to find candidates that fit $P^\star$ closely, because it remains unknown.

The powerful approximators commonly used in SL, like ANNs, happen to be flexible enough to fit every training point in various tortuous ways that resembles nothing the user expects from $P^\star$. This is called overfitting. In this situation, the value of $f_{\mathscr{T}}(\hat{P})$ is high but the only correct prediction of $\hat{P}$ is the training set itself, and any other prediction is aberrant. In the picture labelling problem for instance, $\hat{P}$ perfectly predicts labels for the pictures already labelled, but does only mistakes when given new pictures.

This reveals an implicit *parsimony* hypothesis on the user side: outputs of $P^\star$ should vary somewhat smoothly with the inputs to interpolate the training points, but without adding unnecessary variability between them.

Several approaches in SL aim to alleviate the overfitting problem. One simple approach is to stop the search before the feedback value becomes too high. This prevents overfitting, but requires that the user have a prior idea when to stop.

Alternately, it is common to partition $\mathscr{T}$ into two separate sets: one actual "learning set" $\mathscr{T}_{\mathscr{L}}$ used to evaluate the search feedback $f_{\mathscr{T}_{\mathscr{L}}}$, and one "validation set" $\mathscr{T}_{\mathscr{V}}$ used to assess overfitting once the search has settled on the first. $\hat{P}$ is considered good if it also yields good predictions on the validation set (high $f_{\mathscr{T}_{\mathscr{V}}}(\hat{P})$), because it has better chances not to be overfitting the data points. On the contrary, if it predicts much wrong values on the validation set (low $f_{\mathscr{T}_{\mathscr{V}}}(\hat{P})$), then either it has been overfitting the learning set (high $f_{\mathscr{T}_{\mathscr{L}}}(\hat{P})$) or has not yet been able to fit the learning set at all (low $f_{\mathscr{T}_{\mathscr{L}}}(\hat{P})$). This approach is widely used in SL as it enforces that $\hat{P}$ strikes the right balance between approximate and exact fitting of the training set, and more closely resembles the supposedly "smooth" target $P^\star$.

The above approach comes at the cost that the training data $\mathscr{T}$ must be abundant. There needs to be both enough data for the search to succeed in $\mathscr{T}_{\mathscr{L}}$ and for the validation on $\mathscr{T}_{\mathscr{V}}$ to be meaningful.

As a general trend in SL, the more data in $\mathscr{T}$, the more optimistic the expectations about $\hat{P}$. Increasing the size and diversity of $\mathscr{T}$ also alleviates statistical noise. In fact, there is no upper bound on the size requirement of $\mathscr{T}$ except that, considering that regular calls to $f_{\mathscr{T}}(P)$ are done during the automated search, the processing of $\mathscr{T}$ must remain within reach of computational capacity.

SL is therefore limited by our ability to produce massive training data as humans, while SL agents always crave for more. This has quickly become a major concern of SL and all current SL algorithms are much data-greedy [Hlynsson *et al.* 2019]. It is the subject of ongoing research to determine whether their data-efficiency can still be improved, and at which cost [Cornuéjols, Koriche, *et al.* 2020].

This section has sketched an overview of SL as a ML subdomain. SL on its own is not subject to the problems raised by signature changes, but the question of accommodating these events in SL makes sense when it intersects later with OL and TL (see Figure 2.2).

2.2.2 Unsupervised Learning

UL is another common branch of ML. In this thesis, we do not address signature changes within UL specifically, but UL has useful traditional applications within SL and RL, so we summarize it as an important piece of ML background. UL gathers a broad range of statistical analyses together. In contrast with SL, which essentially boils down to a grouping of phenomenological regression methods, UL refers to very diverse statistical situations, so it is more difficult to summarize in a unified way.

2.2.2.1 UL Principles

Although UL is difficult to unify, the reason that the naming "Unsupervised" opposes the term "Supervised" is consistent across UL. In UL, the agent input data do not have a $(i, o^\star)$ shape like the SL training set $\mathscr{T}$. Instead, the input data, say $\mathscr{D}$, only consist in singletons (i). In the picture labelling example, $\mathscr{D}$ consists in a database with many pictures, but without the labels associated with them. The user goal is to unveil *structure* within this raw dataset. Research questions associated with UL resemble: "Do all the points in $\mathscr{D}$ look the same?" , "Do they fall into several distinct categories?", "Do they line up?", *etc.* In consequence, the signature of the invoked searching agent resembles:

$$\mathscr{D}\ \textit{[raw dataset]} \to A \to \hat{P}\ \textit{[structure within } \mathscr{D}\textit{]}$$

The *dataset* $\mathscr{D}$ is easy to describe: it typically contains a sequence of consistent data "points" i, each described with a predefined list of properties constituting a vector. In the picture labelling example, the properties would be the color values for each pixel. In a dataset representing regular clients of a supermarket, the properties would be the name, gender, age, budget, assiduity, *etc.* of each of them. Consistency of the dataset implies that it basically fits in a rectangular table where each row constitutes a data point and each column represents a property. When all properties are numerical, $\mathscr{D}$ is also typically thought of as raw scatter plot, in an Euclidean space with one dimension per column.

The *structure* $\hat{P}$ sought within $\mathscr{D}$ is more difficult to describe in a unified way, as it ultimately depends on the statistical question at hand. This is where the various types of analyses gathered under the term "UL" start branching. Still, UL analyses have in common that the structure ultimately *simplifies* the data set, in the sense that it makes it easier to describe. In other terms, the unveiling of hidden information within $\mathscr{D}$ makes it less complex to work on, or *compresses* it without loosing much information.

Arguably, in this context, $\hat{P}$ does not resemble the "approximated procedures" seen so far. The unknown structure $P^\star$ is hardly recognized as an "elusive procedure", and some

UL approaches do not exactly perform exploration of a "search space $\mathscr{P}$" to unveil $\hat{P}$ according to the ML principles described in Chapter 1. This makes UL partly stand beyond the frontier between AI and statistical sciences, unless we also argue that the whole "ML" approach to AI is just a fancy name for contemporary *statistics*. Nevertheless, UL is commonly referred to as one major domain of ML. As background to this thesis, we illustrate the diversity underlying UL with 3 different cases in the next section.

2.2.2.2 Various UL Situations

The Clustering Problem This constitutes the most common UL task. In this situation, The question is whether the data points fall into distinct categories, and how to automatically sort these categories out. This is useful for instance in evolutionary sciences, to pack similar genes and species together, or in medicine to spot typical patient immunity response profiles. The signature of P in clustering is described with:

$$i \text{ [data point]} \rightarrow P \rightarrow o \text{ [the category } i \text{ belongs to]}$$

$\hat{P}$ outputs various possible categories. The meaning of a category is that points falling into it look less dissimilar to other points within the same category than to points in other categories. In the picture labelling examples, clustering assembles pictures that share common mean tones. In the supermarket example, it assembles clients with similar profiles.

The user has no further control on the category semantics, unless they enrich the dataset $\mathscr{D}$ with additional properties they wish to see their data discriminated upon. For instance, if the pictures are preprocessed with sophisticated contour detection algorithms, then the categories unveiled by the agent also cluster pictures with similar complexity, fractal dimension, patchwork, *etc.*

Clustering simplifies the original dataset $\mathscr{D}$ in that, instead of describing each object i with all its properties, i is only summarized by the category it belongs to, and the corresponding set of all categories is smaller to describe than the set of all data points.

There exist several clustering approaches. For instance, in "k-means" methods [Lloyd 1982], a distance metric is defined on the input space, and k additional points supposed to represent the categories centroids are appended to the data. The corresponding candidate procedure P is determined by the location of these centroids: P associates each point to the category of its nearest centroid. The algorithm successively updates the centroids locations until it converges to a stable situation $\hat{P}$, where each centroid location corresponds to the mean position of the points associated to it. This algorithm works well in practice, but supposes that the user has a prior idea of the number of categories to look for.

Other approaches to clustering differ in the way the resulting categories are structured. For instance, in "expectation maximization" [Dempster *et al.* 1977], the categories are represented as distinct probability distributions, from which the inputs i have supposedly been drawn, and the essential difference with "k-means" is that categories possibly overlap by blending into each other: *i.e.* one data point i possibly belongs *more* to one

category than another. With "Kohonen maps" [Kohonen 2001], the categories themselves exhibit a particular topology, in the sense that some categories are neighbouring each other while others stand further apart. In the end, the UL user picks one approach or the other depending on how they need to categorize their raw dataset $\mathscr{D}$.

Unveiling Data Distribution In this other common UL task, the challenge is to unveil the data distribution underlying $\mathscr{D}$. In other terms, the user considers that every point in $\mathscr{D}$ is drawn from an unknown stochastic process with probability distribution $P^\star$, and wishes to summarize it with an approximated distribution $\hat{P}$. In this situation, the targeted procedures P constitutes *probability measures* on the input space $\tilde{\mathscr{D}}$ where the data points $\mathscr{D} \subset \tilde{\mathscr{D}}$ live:

$$\mathscr{A} \subset \tilde{\mathscr{D}} \text{ [one region of } \tilde{\mathscr{D}}\text{]} \rightarrow P \rightarrow o \text{ [probability that a data point } i \text{ belongs to } \mathscr{A}\text{]}$$

One common approach to distribution estimation is the "kernel density estimation" [Scott 2018], where the density function of $\hat{P}$ is straightforwardly given by:

$$d_{\hat{P}}(x) = \frac{1}{h\,|\mathscr{D}|} \sum_{i \in \mathscr{D}} K\left(\frac{x-i}{h}\right) \tag{2.13}$$

With K a centered distribution called "kernel", typically a Gaussian density:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{2.14}$$

Binarizing this density function also happens to produce several disjoint categories, which is helpful for *clustering* because it does not require the number of categories to be predefined. Still, the estimation requires that the "smoothing" parameter h be input to the agent, and the binarization threshold also needs to be predefined by the user. Once again, the UL user picks one approach or the other depending on the question at hand.

Principal Component Analysis The last common UL task summarized here is the problem of addressing intrinsic correlations between the input data properties. For instance with the supermarket data, this approach answers questions like: "Are younger clients also the less regular?" or "Is it the case that regular clients are also the ones that spend the most money when they come?". Each such question is easily addressed with one dedicated statistical model, but when the number of dataset properties becomes high, the user wishes a high-level overview of the most related properties.

This is typically achieved with an approach called "Principal Components Analysis" (PCA) [Jolliffe 2011]. PCA consists in a rewriting of all properties in the dataset, or a "base change" of all inputs i coordinates:

$$i \text{ [data point properties]} \rightarrow P \rightarrow o \text{ [data point new properties]}$$

The new properties are carefully crafted so they can be ordered from the one that discriminates all data points best, to the one whose perspective is almost indifferent. This is useful to simplify $\mathscr{D}$ by reducing its dimensionality, since the first few properties approximately summarize the diversity of the whole dataset, while the remaining ones can be discarded without much loss of information.

The transformation is achieved using the algebraic information contained within the covariance matrix of the dataset $\mathscr{D}$. $\hat{P}$ corresponds to the successive projection of data points onto the line spanned by each eigenvector of this matrix, and these projections are ordered with respect to the corresponding eigenvalues magnitude. As a consequence, PCA new properties are linear combination of the others, and the corresponding coefficients of the first dimensions also convey information about the way each original property correlates with the others.

A subsequent limitation of PCA is that only linear combinations can be produced this way. To address non-linear relations among properties, the user needs to explicitly define intermediate, non-linear combinations of properties. In the end, the coefficients of the final combinations are useful for the user to spot highly correlated subsets of the original properties, and dismiss redundant ones as another way of simplifying $\mathscr{D}$.

In summary, UL gathers a range of diverse statistical approaches together. Their common objective is to *analyze* a data set to understand its structure, but without a particular *processing* of this data in mind (*e.g.*, without an expected *outcome* $o^\star$). However, the UL ability to unveil structure within $\mathscr{D}$ makes it typically useful as a methodological component of other ML fields like SL, or RL, addressed in the next section.

2.2.3 Reinforcement Learning

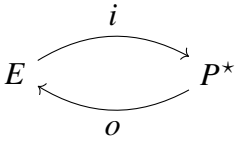
RL is the learning situation adapted to model the example sticky roverbot described in section 1.2, and evolving within its environment.

2.2.3.1 RL Principles

In RL, the "elusive procedure" $P^\star$ is not expected to be useful as a single, one-shot process like in:

$$i \rightarrow P^\star \rightarrow o$$

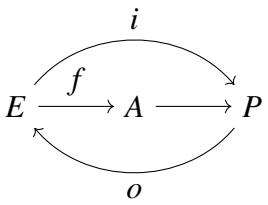
Instead, the true "behaviour" sought by the user emerges from the continuous interaction between $P^\star$ and its surrounding *environment* E:



In other terms, the assessment of P depends on the many successive values of o outputted by the process in reaction to E, so RL implies that *time* be a major component of the learning context. The roverbot example is a machine supposed to follow a user anywhere: i constitutes the data provided by the bot sensors (*e.g.*, user direction and distance) and o is the data sent to the bot actuators (*e.g.*, steer and speed control). $P^\star$ is the ideal internal procedure supposed to correctly compute outputs to inputs so the robot task is fulfilled (*e.g.*, the robot program), and E represents everything outside of the robot decisional scope (*e.g.*, user, weather, ground quality, hardware quality, ...). In this situation, the agent has to continuously react to its surroundings to keep exhibiting the right behaviour.

RL is also the appropriate learning context for other famous AI problems, like playing human games [Mnih, Kavukcuoglu, Silver, Graves, *et al.* 2013; Mnih, Kavukcuoglu, Silver, Rusu, *et al.* 2015; Silver *et al.* 2016; Berner *et al.* 2019], or processing human speech [K.-F. Lee and Mahajan 1990; Goddeau and Pineau 2000]. In this context, the behaviours P are commonly referred to as the possible agent's *policies*, the inputs i as the perceived environmental *state*, and the outputs o as the *actions* resulting from the current policy [Sutton and Barto 2018].

Consistently with SL, a behavioural search space $\mathscr{P}$ is defined, and the agent explores $\mathscr{P}$ to find a good policy $\hat{P}$. A crux difference with SL is that there is no *training set* available to the agent. Because of the time component of RL, $P^\star$ can only be expressed as a high-level expectation, *e.g.* "stick to user", which typically spans among multiple time steps. The user has no particular idea what to output as successive low-level $o^\star$ values to achieve it, so there is no prior $(i,\ o^\star)$ "training" data to use as hint. Instead, the agent is supervised with a high-level feedback signal f (*e.g.*, proximity to user) assessing, from the user perspective, whether it is currently doing well:



This is the reason for the most challenging RL-specific problem: since the feedback signal f is evaluated on a higher time scale, there is a noticeable decoupling between the individual decisions of the agent and their actual repercussions on f.

For instance with the sticky bot, succeeding in getting 1 meter closer from the user implies that many intermediate correct decisions have been taken for a while: *e.g.*, steering right, run towards user, avoid obstacle, resume correct direction. All these actions have been taken without information whether they were successful or not. And when the reward finally comes under the form of a high feedback value, it is unclear to the machine which (if any) of all these past actions should be the most credited for success,

and therefore *reinforced*. This is known as the *credit assignment* problem [Watkins 1989; Sutton and Barto 2018]. To alleviate it, RL methods commonly keep track of past decisions, and attempt to estimate how important they were in the process.

RL distinguishes between two different kinds of tasks. In *episodic* tasks, the agent-environment retroaction loop has an expected termination point. For instance, if the roverbot mission is to escape from a maze, the goal is achieved as soon as the exit is reached. These tasks have to be replayed numerous time before the agent figures out how to solve them best. In *continuing* tasks, on the other hand, there is no expected termination point. This is the case with the roverbot whose mission is to follow the user indefinitely. The task keeps going while the agent progressively improves. Although these two kinds of tasks constitute qualitatively different learning situations, the RL framework makes it possible to reason about both in mostly the same terms [Sutton and Barto 2018].

2.2.3.2 RL Methods and Limits

There are two major types of approaches to finding $\hat{P}$ in RL, They relate to the way $\mathscr{P}$ is constructed.

In *value-based* approaches, one objective of the agent is to construct a reliable estimation of the future expected feedback associated with each possible decision. This association is called "value function" and constitutes a biased model of the environment. The model is biased because it only represents environmental dynamics from the perspective of the feedback they eventually provide to the agent, and any external mechanics indifferent to the feedback does not belong to the value function. Based on this estimation, the optimal procedure $\hat{P}$ is defined as the trivial procedure always picking the highest-valued decision. As a result, $\mathscr{P}$ boils down to the space of all possible value functions. This is the case of methods known as SARSA, TD(λ), or Q-Learning [Sutton and Barto 2018].

$\hat{P}$ is trivial to compute given a correct value function, but the value function is sometimes more difficult to approximate than the very targeted objective behaviour $P^\star$, because it implies that the agent constructs an understanding of "how the environment works" instead of just "what it should do within in". In alternate, *policy-gradient* based approaches, the value function is not explicitly targeted, and $\mathscr{P}$ is simply defined as the space of possible $i \to o$ associations. The procedure $\hat{P}$ eventually retained is one yielding consistently good feedbacks, regardless the external mechanics it triggers. This is the case of methods known as Actor-Critic or REINFORCE [Sutton and Barto 2018].

In simple, scholar RL cases commonly referred to as *tabular cases*, there is a small, finite number of possible combinations of input/output pairs $(i,\ o)$. This provides strong guarantees that the credit assignment problem is overcome by the above methods, and even that the optimal behaviour $P^\star$ is eventually reached exactly.

However, tabular cases fail to represent many real-world situations where the various possible inputs and outputs are intractable: either they are too numerous (*e.g.*, possible situations on a Go board), or they are infinite (*e.g.*, text messages) and even continuous

(*e.g.*, user direction and distance) [Rachelson *et al.* 2008; Sigaud and Stulp 2019]. In such cases, the agent needs to perform approximations of the environmental process E along with approximations of the space of possible policies $\mathscr{P}$, a typical use case for ANNs.

In traditional applications of RL, it is also commonly assumed that the environment E is a Markovian process, either stochastic or deterministic, with no memory of past events. Under this assumption, the above methods guarantee that an optimal approximated behavior $\hat{P}$ is eventually reached, which alleviates the credit assignment problem.

However, the environment surrounding the task does exhibit a memory in many real-world situations. With the sticky roverbot for instance, the location of an obstacle differs whether the bot has bumped against it in the past or not. In these cases, not only the current environmental state and the current agent decision are relevant to the next iteration, but also states and decisions arbitrarily distant in the past. E is still considered Markovian, but it is then assumed to contain traces of past events commonly represented with inner variables called *hidden states*.

Hidden states legitimately constitute the environment, but the agent gets no information about them in the inputs. In this situation, although the above methods are still useful in practice, their guarantees do not hold anymore. Unveiling adaptations of RL methods that best accommodate the problem of intractable input/output spaces and the problem of environmental hidden states is still the object of ongoing research.

The inner architecture of RL agents is commonly adapted depending on the challenge at hand. For instance, it is either "monolithic" with *e.g.*, one end-to-end ANN supposed to directly connect inputs to outputs, or "modular", with a more sophisticated internal structure involving distinct components interacting together. Modular approaches are motivated by various possible expectations. First, the user possibly seeks a *hierarchical* representation of P, with a decoupling of high-level and low-level policies, so as to improve performance or ease transfer [Ring 1994; 1997; Partalas *et al.* 2008]. See also [Devin *et al.* 2017], [Frans *et al.* 2018], [Clavera *et al.* 2017] and [Suro 2020] for applications of hierarchical approaches in robots. Second, the user possibly seeks *coordination* between multiple agents, see [Ono and Fukumoto 1996; Pu-Cheng *et al.* 2006; Partalas *et al.* 2008] for examples in 2D grids, and [Uchibe *et al.* 1996] for an example in soccer robots. Modular architecture is also a way to inject *priors* into the learning procedure, which constitutes a shift towards more mechanistic approaches. This is used in [Sprague and Ballard 2003] to improve on multi-objective tasks, in [Kalmár *et al.* 1998] for grabbing robots, in [Hanna *et al.* 2010] to reduce state dimensionality in games and in [Jin *et al.* 2012] for traffic optimization.

In summary, the RL learning context has proven to be flexible enough to address many practical problems. This thesis attempts to extend its application range further by featuring situations undergoing signature changes, as described in section 1.2. This said, the problem of signature changes is not restricted to RL, as there also exist orthogonal learning situations (see Figure 2.2). These are described in the next section.

2.2.4 Transversal Learning Contexts

The learning situations constituting SL, UL and RL differ much, which makes these three fields of ML essentially disjoint. However, they are connected in various ways. For instance, UL or SL approaches are typically used as technical components within RL methods: *e.g.*, UL is useful to compress and simplify the input space of RL agents, SL is useful to approximate the value function in value-based methods [Levine and Koltun 2013], or the environment itself. As such, any progress in either is also beneficial to RL.

There also exist higher-level problems defining learning situations that orthogonally happen in SL, UL and RL. Therefore, they converge in their approaches to face them. This is the case of Online Learning (OL) and Transfer Learning (TL).

2.2.4.1 Online Learning

The exploration of $\mathscr{P}$, like the heuristics defined in Section 2.1.2, is typically an iterative process involving multiple sequential steps.

This implies that there is a *timeline* associated with the search process. On each time step, one candidate procedure P is drawn. Evaluation of P by the feedback signal f indicates whether this candidate should be saved as a promising candidate for $\hat{P}$, or if it should be dismissed to test another P on the next step. This timeline is internal to the agent.

On the other hand, there is the user timeline. Time steps along this timeline correspond to events external to the agent. In SL or UL for instance, the user first needs to gather $\mathscr{T}$ or $\mathscr{D}$ input data, then they run the search agent and then finally collect the results. All this constitutes one time step.

In a nutshell, the *agent* timeline is the one over which the automated ML search is performed (one time step corresponds to the testing of one candidate P) while the *user* timeline is the one over which the input data is produced (one time step corresponds to new input data availability). As such, we also refer to them as the *search* timeline and the *input* timeline.

The input timeline is trivial in the plain SL or UL cases, because in only contains one time step. But interestingly, the two timelines happen to line up in the RL situation. In RL, the agent iteratively receives inputs i and f, so the input timeline contains many steps. In addition, no improvement of the current candidate procedure P can be done on the agent side as long as no new input data is produced on the user side. In other terms, the agent search *blocks* on each input step, as it needs more data to keep going: the two timelines are nested into each other.

Note that the two timelines do not necessarily step at the same pace: *e.g.*, the agent possibly waits for a few input steps before updating the search. Alternately, the agent possibly performs several search steps before waiting for new input data to be available. Nevertheless, the two timelines are locked within each other due to the learning configuration. This situation is commonly referred to as a case of *incremental* learning, or Online Learning (OL) [Jain *et al.* 2014; Losing *et al.* 2018].

RL constitutes a natural OL situation. For this reason, it is non-trivial to use RL algorithms to take advantage of "offline" contexts where the whole training data is readily available at once, but see [Levine, Kumar, *et al.* 2020] for recent insights into bridging this gap. However, OL also extends outside the strict scope of RL: there are cases of OL in SL (OSL) [Gama *et al.* 2014] and in UL (OUL) [Cui *et al.* 2016]. For instance, in the picture labelling example, the training data is long, costly, and slow to produce, because humans need to label many pictures by hand. Instead of waiting for the training set $\mathscr{T}$ to be complete, the user prefers that a rough automated procedure $\hat{P}$ be ready as soon as the manual labelling starts so as to use it right away, should this procedure be subsequently improved as new manually labelled examples $(i,\ o^\star)$ are made available. In this situation, the SL agent search becomes blocked on every input step, waiting for new labelled examples to progress further. Another reason to stream the input data is that they are too big to be processed as a whole by the learning agent, or because they are produced faster than they can actually be stored [Reinsel *et al.* 2018]. This situation constitutes an iterative learning process like:

$$(i,\ o^\star) \textit{ [new labelled picture]} \rightarrow A \rightarrow \hat{P} \textit{ [best candidate found so far]}$$
$$(i,\ o^\star) \textit{ [new labelled picture]} \rightarrow A \rightarrow \hat{P} \textit{ [best candidate found so far]}$$
$$\ldots$$

A simple approach to OSL is to restart the learning from scratch on each new input. But the exploration of $\mathscr{P}$ is costly so the user rather wishes that the search be resumed from its latest state as soon as a new input is made available. As a consequence, the input timeline and the search timeline become locked together again, and the learning situation becomes an OSL problem. A similar situation is possible with UL: OUL happens when the raw dataset input $\mathscr{D}$ is a data stream that the agent needs to accommodate whenever it updates. Between two successive updates, the agent internal search steps freeze until the next piece of data becomes available.

OL sometimes demands that major technical aspects of the internal agent process be adapted, so as to regularly stop and resume the search. Regarding this, the major concern of OL is to balance computing efficiency of each time step with the quality of every yielded, temporary approximation $\hat{P}$.

One typical OL challenge is that the distribution of incoming inputs sometimes varies over the input timeline. For instance, in the picture labelling example, the agent initially receives photos, so it starts progressing towards an approximated procedure $\hat{P}$ that correctly processes photos into their matching labels. But at some point, the incoming data stream starts yielding drawings and cartoons, so the procedure needs to be adapted to also feature them. This results in that the targeted approximation $\hat{P}$ becomes a moving target for search procedure, a situation commonly referred to as "Concept Drift" [Widmer and

Kubat 1996; Tsymbal 2004; Gama *et al.* 2014; Heng Wang and Abraham 2015; Webb *et al.* 2017; Lecarpentier and Rachelson 2019]. Other possible modifications occurring during the course of learning are the *signature change* studied in this thesis. For instance, the two PL experiments exposed in Chapters 6 (OSL) and Chapter 7 (RL) happen in an OL context.

Concept drifts and signature changes reveal that the very notion of "task" is not always straightforwardly defined, either from the user or the agent perspective, and even occasionally changes over time. Making agents seamlessly handle changing tasks is the core concern of another transversal field of ML called Transfer Learning (TL).

2.2.4.2 Transfer Learning

Exploring the search space $\mathscr{P}$ is the principal activity of ML agents, and is computationally costly. The major outcome of the search process is the approximated procedure $\hat{P}$, sometimes referred to as the "knowledge" unveiled by the agent.

To alleviate the cost of learning, ML users notice connections between different learning tasks, because some of them look more similar to each other. For instance, consider an automated agent whose task is to discriminate photos taken by night from photos taken during daytime. Chances are that it partly works the same way as the former agent distinguishing kittens from planes, *etc.* As such, the user expects that some of the knowledge learned by the agent in the former task, called the *source* task can be reused in the new task, called the *target* task, so that the second learning phase is less costly:

$$\mathscr{T} \text{ [training set]} \to A \to \hat{P}_{source} \text{ [discriminates kittens]}$$
$$\left(\mathscr{T} \text{ [new training set]}, \hat{P}_{source}\right) \to A \to \hat{P}_{target} \text{ [discriminates daytime]}$$

Note that the search space possibly needs to be upgraded from one task to the other. In other terms, $\mathscr{P}_{source}$ and $\mathscr{P}_{target}$ possibly differ, so a mapping between the two has to be constructed to *project* $\mathscr{P}_{source}$ into $\mathscr{P}_{target}$. This situation is referred to as Transfer Learning (TL).

The trivial, naive approach to TL is to just ignore previous knowledge $\hat{P}_{source}$ and start the target learning from scratch as if the two tasks were unrelated. In this naive approach, the target learning phase is costly, whereas it possibly performs a redundant exploration compared to the source learning. Instead, clever TL techniques attempt to extract some of the "knowledge" contained within $\hat{P}_{source}$ and *transfer* it into the target task to make the target search more efficient.

Like OL, TL is a higher-level learning situation that seamlessly occurs within SL, UL or RL. In any case, the aim is that the TL agent be more efficient than the naive agent. The *signature change* problem addressed in this thesis falls exactly within the scope of TL, and the *projection* operation supposed to connect $\mathscr{P}_{source}$ to $\mathscr{P}_{target}$ is the core corresponding technique, defined in Chapter 5 then tested in Chapter 6 and Chapter 7.

TL is challenging, and very few analytic approaches provide theoretical guarantees in this field [Taylor and Stone 2009]. One first challenge is to evaluate the quality of the transfer, because there are various ways it is possibly beneficial.

First, the *initial performance* of the agent on the target task is possibly better than the initial performance of the naive agent. This is known as the *jumpstart* benefit, and happens because some progress towards $\hat{P}_{target}$ has already started during source learning. Second, the agent possibly *learns faster* on the target task, because it inherits from some form of intimacy with the search space. It therefore reaches $\hat{P}_{target}$ sooner. Third, the *final performance* of the agent on the target task is possibly better than the performance of the naive agent. This happens for instance if spurious candidates procedures (*e.g.*, local optima) have already been avoided and ruled out during source learning.

Depending on the situation, the TL agent benefits from either a combination of these advantages, or from none, so they are difficult to measure and distinguish from each other. It is the object of ongoing research to define standard metrics for measuring TL advantage [Lopez-Paz and Ranzato 2017].

The major concern of TL is known as *negative transfer*. This phenomenon makes the TL agent *less efficient* at solving the target task than the naive agent.

Negative transfer possibly happens when the source and target approximated procedures $\hat{P}_{source}$ and $\hat{P}_{target}$ differ so much that very few parts of $\hat{P}_{source}$ can actually be reused in $\hat{P}_{target}$. In this situation, the TL agent needs to *unlearn* source knowledge before it eventually finds an acceptable target procedure, which makes it less efficient than a fresh, naive agent. In other terms, negative transfer occurs when $\hat{P}_{source}$ turns out to be more of a burden than a useful hint to the agent.

As $\hat{P}_{target}$ is unknown before the transfer, it is difficult to assess whether or not negative transfer is likely to occur, and there is no guarantee against this phenomenon in the general case. Preventing negative transfer, or predicting it to avoid using TL techniques in situations where it would be a burden, is still the object of ongoing research.

In some TL situations, the user expects that the end agent be able to eventually solve *both* the target task and the source task. This implies that $\hat{P}_{target}$ is an extension of $\hat{P}_{source}$, so $\mathscr{P}_{source}$ is contained or injected within $\mathscr{P}_{target}$. A common issue in this situation is known as *catastrophic forgetting*, a phenomenon where the agent performance on the source task is actually degraded as it improves on the target task.

For instance in the picture labelling problem, the more the agent is able to distinguish daylight pictures from night pictures, the more mistake it makes when addressing whether the pictures contains planes or kittens. Maybe the reason is that forgetting knowledge is sometimes necessary to adapt new tasks, but this is still unsure to ML science, and the right balance between stability and plasticity of the learning agent still needs to be struck.

For instance, to avoid catastrophic forgetting, some approaches focus on transferring higher-level "knowledge" like technical components of the agent search procedure itself:

e.g., learning rates in [Finn *et al.* 2017], hierarchical layers in [Ring 1994; 1997], or composed actions in [Bacon *et al.* 2017; Frans *et al.* 2018], instead of the direct source procedure $\hat{P}_{source}$. In [Rusu *et al.* 2016] an ANN is designed to capitalize on past experience without ever forgetting, at the cost of ever-growing needs in resource. In [LeCun *et al.* 1990; J. Kirkpatrick *et al.* 2017; Zenke *et al.* 2017; Kaplanis *et al.* 2018], ANNs are gifted with a certain learning *capacity*, and learn several new tasks without altering performances on previously encountered ones, until their learning capacity becomes saturated.

Note that, while negative transfer is always undesirable because it cancels the benefits of TL, catastrophic forgetting is only undesirable in specific situations where past learning contexts are expected to be faced again. As a consequence, TL agents designers always attempt to avoid negative transfer, but they have a choice how much computing resource to allocate into alleviating catastrophic forgetting, depending on the use case.

The Protean Learning (PL) situation exposed in this thesis is rigorously defined in Chapter 5 as an instance of TL (see Figure 2.2). Accordingly, the connections between PL and TL is described more in depth in Chapter 4. In Chapters 6 and 7, we conduct experiments addressing the general negative transfer problem during signature changes in both OSL and RL, leaving the particular problem of catastrophic forgetting for future dedicated works.

Chapter 3

Problem Statement

In the previous two chapters, we have familiarized ourselves with computer learning procedures, with the problem of live signature change and with general relevant background in the domain of ML. The remainder of this manuscript is specifically focused on signature changes and Protean Learning (PL). This chapter states the very subject of the thesis. First, we wrap up the problems related with signature change that we have encountered so far. Then, we formulate our core position regarding signature changes, that we defend in the thesis. In particular, we focus on addressing $(+i)$ and $(-i)$ events more specifically.

3.1 The Signature Change Problem

Let us first summarize the learning situations involving signature changes that we have seen so far. In Section 1.2, with the help of an imaginary roverbot example, we have explained the various challenging situations faced when the expected signature of the agent internal procedures undergoes changes during the course of learning.

We have seen that these changes come in several flavours. First, new inputs can be fed into the agent with events like $(+i)$ (input addition) or $(+f)$ (feedback addition). Conversely, inputs can also be removed from the agent with events like $(-i)$ (input deletion) or $(-f)$ (feedback deletion). On the other hand, new outputs are expected from the agent when $(+o)$ (output addition) events occur, and have to be dismissed on $(-o)$ (output deletion) events. All these events are distinct from $(\sim E)$ (environmental change).

As noted before, there is a fundamental difference between events $(+i,\ -i,\ +o,\ -o)$ and $(+f,\ -f)$. The former induce changes in the input/output signature of the agent inner behavioural procedure P, while the latter induce changes in the input signature of the higher-level agent search procedures A and S instead, whose output is P itself. Within the scope of this thesis, we consider that no extra procedure is ever expected from the agent as the result of a signature change event (no "$(+P)$" or "$(-P)$" events).

There also exist softer kinds of signature change events, noted $(\sim i)$ (input change), $(\sim o)$ (output change), and $(\sim f)$ (feedback change). When these occur, no data stream is added to or withdrawn from the agent interface, but the range of values they possibly carry

is modified. For instance, $(\sim i)$ happens when a thermometer sensor widens its sensitivity range, resulting in that the agent has to process values never encountered before, or when it restricts it, resulting in that some values will never be encountered again. This particular kind of events is further discussed in Section 5.1.

Signature changes do not only occur in modular robots like the roverbot example. They are more general, and naturally occur in any long-term OL procedures, when every learning step is subordinate to the progression of external input data streams. For instance, $(-i)$ occurs any time an input stream fails, because the data source has become extinct or obsolete, and $(+i)$ occurs any time a new input stream is considered relevant by the agent designers. Section 1.2 has already considered automatic trading learners as an example long-term OL procedure, but signature changes also occur in modular and developmental robotics [Ababsa *et al.* 2014; Doncieux *et al.* 2015], or in adaptive games [Francillette 2014; Bonnici *et al.* 2019].

In general, the search space $\mathscr{P}$ containing every candidate behavioural procedures P is *transformed* when signature changes occur, so the learning process is now exploring an outdated landscape. This is what we call *signature change problem*. For instance, when $(-i)$ occurs in a RNN-based RL learner, there is no enough input data to compute outputs of the formula given by the inner neural network, so the program mandatorily blocks if no accommodating operation is performed. This failure is more serious than a lack of convergence, or poor performances on the task at hand: after $(-i)$, the agent procedure ceases being defined because of wrong procedures arity, so it cannot be run by a computer anymore.

In Chapter 2, we have made a tour of ML as a computer science discipline, and seen how SL, UL, RL, OL or TL structure the research domain (see Figure 2.2). As a matter of fact, most approaches to ML in these disciplines work under the implicit assumption that the signatures of the learning agent and its inner procedure are *fixed* throughout the learning process, so they do not tackle the signature change problem. Consequently, accommodating signature changes in PL cannot consist in trivially applying existing ML methods.

In other terms, although PL constitutes an essential learning context, it appears that signature changes *per se* are not addressed yet by the ML community. This thesis attempts to engage into this domain, with a special focus on $(+i)$ and $(-i)$.

3.2 Thesis

In this thesis, we defend that Protean Learning (PL) constitutes a significant, new and non-trivial domain of ML. PL is significant because of the transient nature of the data streams processed by computer procedures, resulting in that signature changes occur in common natural tasks targeted by ML. PL is new because it explicitly focus on signature changes, whereas the learner signature is mostly considered constant by contemporary ML approaches.

Regarding triviality, PL inherits from the triviality of TL in general. When a change event occurs during the course of learning, and the agent is not able to handle the new situation, there is always one trivial solution to accommodate the change. The trivial solution is to define a new, fresh adjusted agent, for example an agent with correct arity. This agent is called *naive*, and commences a novel learning session from scratch in the new situation, until it eventually solves the task again. This solution is trivial because it only requires that the ML methods used to train the former agent (*e.g.*, before the signature change) be used again to train the naive agent (after the signature change), so it involves no particular groundwork.

However, the trivial solution requires that much computing resources (time, power, development, *etc.*) be allocated again so the naive learner fulfills its fresh training. The more complex the task at hand, the more costly this new training phase. The cost is justifiable if the new task is much different than the original one, but there are chances that it is unnecessary if the change is slight. When the change is slight, the naive agent spends much of the learning process to recover skills that the former agent already had, so the computation is redundant.

To avoid this, the idea of TL, and of PL in particular, is to *transfer* the abilities of the former agent to a new adjusted agent, so it does not have to learn everything again. This factorization of the learning effort is useful not only to save a few immediate resources, but to save them repeatedly if the changes are small and numerous, or even gradual.

In the end, the goal of PL and TL is not exactly to accommodate the change or succeed in the resulting task — because this is easily done by the trivial solution — but to improve the performance of this accommodation with respect to the corresponding naive agent. This requires non-trivial groundwork, as it depends on insights into the learning method used, on the task at hand and on the nature of the change. For instance, a good transfer can be realized if the change is slight, but no performance gain can be achieved when the new task is so different that there is no learning effort to factorize at all. In such cases, the trivial solution is already optimal.

Regarding $(+i)$ and $(-i)$, our positioning is more precise and has stronger implications. In the remainder of the thesis, we focus on a protean agent A_{p_1} needing to accommodate one particular signature change event so it becomes A_{p_2} with a transfer technique (see Figure 5.3). In contrast, the naive agent A_n restarts the learning over from scratch. Note that A_{p_2} and A_n have the same signature and are trained on the same task. In the end, PL is considered a success if A_{p_2} yields better performances than A_n on the task. Ensuring this requires that the change event and the current learning method be known, but also possibly requires deep insights into the particular task.

As we have seen in Chapter 2, the strength of ML approaches and associated techniques is that they are sufficiently generic to be applied in a variety of learning situations, regardless of the particular task. As a consequence, it would be undesirable that PL required strong *ad hoc* knowledge of the task. In this thesis, we defend that it is

not the case. Although a strong *ad hoc* knowledge of the task at hand would indeed enable optimal and finely-tuned accommodation of signature change events, we defend that there also exist simple generic operations, referred to as *projections*, able to transform A_{p_1} into A_{p_2} without requiring particular knowledge of the task at hand, and still yielding better results than the trivial solution A_n on average.

To this end, we contribute to the uncovering of PL as a subdomain of ML with three contributions in this thesis:

- Chapter 5 provides a formal specification of PL, along with a precise definition of a "signature change" (any variation of i^Δ, o^Δ or f^Δ) and the "task at hand" (E). We also restrict our focus to two particular change events: input addition $(+i)$ and input deletion $(-i)$, which constitute the basis of Input-related PL (IPL). The generic IPL projections associated to these events are rigorously specified (the "natural projection" for $(+i)$ and the class of "almost-natural projections" for $(-i)$).

- Chapter 6 addresses these particular two IPL events in the context of OSL. With a carefully controlled synthetic experiment, we show that a simple application of the generic projections on a RNN-based learner yields A_{p_2} agents that outperform A_n in this context. In addition, we qualify this advantage depending on various properties of the task, which addresses the strengths and limits of the generic IPL approach. These results have been published in [Bonnici *et al.* 2020].

- Chapter 7 addresses the same particular two IPL events in the context of RL. The intent is to port the previous results into a more sophisticated online learning situation featuring the credit assignment problem. To this end, a formal benchmark, constituted of numerous tabular RL environments dedicated to basic assessment of IPL, is specified and analyzed. Then, two traditional RL learners (Q-Learning and Actor-Critic) are tested against this benchmark as they undergo $(+i)$ and $(-i)$ events. Again, we show that a simple application of the generic projections to these learners yields A_{p_2} agents that outperform A_n on average. And we qualify the advantage of IPL depending on various properties of the environments in the benchmark.

In the end, we conclude that PL is not only interesting because it relates to challenging unaddressed natural learning situations, but also because there exist simple generic PL approaches, at least in the case of IPL events, that correctly face these challenges at least in OSL and RL contexts. For this reason, we suggest that PL be further explored as a domain of ML, and that alternate generic projections corresponding to other events like $(+o,\ -o,\ \sim o,\ +f,\ -f,\ \sim f,\ \textit{etc.})$ be investigated in future works.

Chapter 4

State of the Art

This chapter summarizes various works related to the problem of accommodating signature changes in ML. We explain how they are connected to PL, or not, so it is easier to understand our positioning in the next. We conclude the chapter with an overview of future PL milestones.

4.1 Documentation Method

Our bibliographic research in the restricted area of PL spawned from three major reviews that structure the relevant literature. First, the book by Sutton and Barto 2018 (second edition) provides a deep insight into RL and a thorough overview of the state of the art in this domain, listing both various contemporary approaches to RL and unaddressed problems. Then, the review by Taylor and Stone 2009 provides a deep analysis of the interplay between RL and TL, while the chapter by Torrey and Shavlik 2010 addresses TL in general. They were very useful to investigate how PL fits into the picture. Starting from these points, publications were crawled up to most recent works, including the review from Losing *et al.* 2018 about incremental approaches in OSL.

In this manuscript, considering that PL is a particular subdomain of TL, we only included works that investigate TL, either within SL, UL or RL, and regardless of whether they explicitly mentioned the notion of signature. Works with only a focus on learning performances outside any contextual change were excluded.

Related works were compared according to various criteria listed in columns of Table 4.1. For example, we distinguished whether the addressed change was a simple environmental change ($\sim E$) or a signature change ($+i$, $-o$, $\sim f$, *etc.*). We also distinguished whether the change is supposed to occur during the course of learning (so it relates to OL) or between different learning sessions.

To clarify our positioning, two lines referring to PL are added at the bottom of the table. The first refers to PL in its broadest intent (see Section 5.1), addressing every signature change in any context, but leaving plain environmental changes to generic TL approaches. The second corresponds to the restricted part of PL specifically focused on.

Related Works	$(\sim E)$	i	o	f	OL	TL	CF
Heterog. TL [142]	×	✓ $(\sim i)$	×	×	× (UL)	**(1)**	×
Heterog. DA [76]	×	✓ $(\sim i)$	×	×	× (SL)	**(1)**	✓
Open-Set DA [14]	×	×	✓ $(\sim o)$	×	× (SL)	×	×
Concept Drift [138]	✓	×	×	✓ $(\sim f)$	✓ (OSL)	**(4)**	✓
Dynamic Evolutionary Optimization [106]	✓	×	×	✓ $(\sim f)$	✓ (OSL)	**(4)**	×
Successor Features + Gen. Pol. Imp. [8][7]	×	×	×	✓ $(+f)$ $(-f)$	✓ (RL)	**(2)**	✓
Gradient Episodic Memory [82]	✓	×	×	×	× (SL)	**(4)**	✓
Distral [129]	✓	×	×	✓ $(\sim f)$	✓ (RL)	**(3)**	×
Modular Policies for Multi-Task Transfer [27]	✓	×	×	✓ $(\sim f)$	✓ (RL)	**(2)**	×
Curric. Learning [9]	✓	×	×	✓ $(\sim f)$	× (SL)	**(2)**	×
Multiple Outlooks [45]	×	✓ $(+i)$	×	×	× (SL)	**(3)**	✓
Progr. Networks [107]	✓	×	×	✓ $(\sim f)$	✓ (RL)	**(4)**	✓
Protean Learning (broadest sense)	×	$(+i)$ ✓ $(-i)$ $(\sim i)$	$(+o)$ ✓ $(-o)$ $(\sim o)$	$(+f)$ ✓ $(-f)$ $(\sim f)$	✓ (OL) × (~~OL~~)	**(1)** **(2)** **(3)** **(4)**	✓
Input-PL (in this thesis)	×	✓ $(+i)$ $(-i)$	×	×	✓ (OSL) (RL)	**(4)**	×

Table 4.1: Overview of previous works related to accommodating changes during learning. Numbers in brackets refer to bibliographic entries (page 179). Numbers in parentheses refer to TL situations described in Section 4.2.2. Tick marks summarize characteristic features of the approaches:
$(\sim E)$: Is the approach concerned with environmental changes?
i / o / f: Is it concerned with input / output / feedback signature changes?
OL: Does the change occur during the course of learning?
TL: Which transfer situation is it addressing?
CF: Does it attempt to alleviate Catastrophic Forgetting?

4.2 Related Works

The works listed in Table 4.1, and the comparison criteria corresponding to the last two columns, are described in this section. First, the overview of TL sketched in Section 2.2.4.2 is fleshed up with additional details regarding the motivations and the challenges in this field. Connections to other fields in relation to PL are also drawn, along with an original taxonomy of various TL situations.

4.2.1 The Motivation for Transfer Learning

The domain of Transfer Learning (TL) defines a learning situation that strongly relates to PL. In TL, the agent has already found acceptable solutions to a set of tasks called

"source" tasks, so it is supposed to carry some kind of "knowledge". The objective is to benefit from this knowledge while tackling a new task called a "target" task. The knowledge is said to be "transferred" from the source task to the target task. In other words, the TL agent is expected to generalize not only *within* tasks, but also *across* tasks [Taylor and Stone 2009; Torrey and Shavlik 2010]. Signature changes do constitute a modification of the task at hand, so PL must also be considered an opportunity to "transfer" knowledge from a "source" task (before the change event) to a "target" task (after the change event).

From a general perspective, the core motivation of TL is to improve learning efficiency on the target task. In terms of our particular problem statement (see Section 3.2): if the source agent A_{p_1} has its knowledge transferred into target agent A_{p_2}, then the objective is that A_{p_2} performance be better than the performance of a naive agent A_n learning from scratch on the target task. There are various ways that A_{p_2} can be better than A_n, depending on the situation, which better qualifies the possible advantages of TL:

- Possibly, the *initial performance* of A_{p_2} is better than the initial performance of A_n. This is known as the *jumpstart* benefit, and happens because the convergence process has already started during source learning: A_{p_2} starts behavioural exploration from approximately the ending point of A_{p_1}, which is likely better than the naive starting point of A_n.

- Possibly, A_{p_2} *learns faster* than A_n, because transfer from A_{p_1} makes it inherit from some form of intimacy with the task at hand. It therefore reaches a successful state sooner.

- Possibly, the *final performance* of A_{p_2} is better than the final performance of A_n. This happens for instance if dead-end local optima have already been avoided by A_{p_1} and ruled out during source learning, so there are maximization traps that A_{p_2} has escaped but A_n got stuck into.

Note that a good TL agent possibly cumulates several of the above advantages, which makes it possible to outperform naive agents and save a lot of computing resources. For this reason, TL is investigated in numerous domains of ML. As an instance of TL, PL agents also aim to benefit from the above advantages when accommodating signature changes.

4.2.2 The Various Transfer Learning Situations

As a research domain, TL is transversal to ML and applies to UL, SL and RL. Relevant literature at the intersection between these domains is summarized below, and a few key publications are reported in lines of Table 4.1 to clarify the positioning of PL with respect to previous works.

In UL for instance, Yang *et al.* 2009 feature transfer between heterogeneous data sources to improve input flexibility of an image clustering algorithm. This approach is

related to "Heterogeneous Domain Adaptation" [Li *et al.* 2014], a research domain resembling PL in that the *input* interface of learning agents is supposed to widen its range of possible source data. In [Busto and Gall 2017], an approach called "Open-Set Domain Adaptation" also attempts to make the *output* interface of learning agents more flexible, although it does not feature an explicit transfer between two tasks.

In more traditional SL contexts, some foundational work in TL is set by Thrun 1995 using classification problems. Thrun suggests that TL is a key condition to ambitious long-lived learning agents supposed to meet various numerous tasks in sequence, a learning context also referred to as "Continuous Learning" or "Lifelong Learning", and also founded in [Ring 1994; 1997]. In [Lecarpentier and Rachelson 2019] for instance, non-stationary RL environments undergo continuous changes.

In [Widmer and Kubat 1996], the authors also attempt to accommodate an important phenomenon known as "Concept Drift", resulting in that the distribution of input data and the environmental context change over the course of learning, necessitating continuous transfer. This problem is reviewed in [Tsymbal 2004], and in [Gama *et al.* 2014] in the restricted scope of OSL. See also [Žliobaitė *et al.* 2016] for a review of applications in SL. In [Heng Wang and Abraham 2015], the phenomenon is detected as a noisy signal. In [Rohlfshagen *et al.* 2009], the corresponding problem is also tackled with formal approaches. In [Jaber *et al.* 2013], it is questioned which outdated information to drop first. However, the signature of the learning agent is assumed to be fixed in these works.

Regarding RL, Taylor and Stone 2009 and Lazaric 2012 offer extensive reviews of the interplay between RL and TL. In [Barreto, Munos, *et al.* 2017] then [Barreto, Borsa, *et al.* 2018] for instance, an approach called "Successor Features", combined with "Generalized Policy Improvement", is developed to transfer RL knowledge across tasks that differ by their reward functions. In [Teh *et al.* 2017], the knowledge is "distilled" with a method called "Distral" so that transfer works across parallel RL tasks.

Throughout the literature, and beyond the strict distinction between SL, RL and UL, we found that there are various different situations in which TL is invoked, but it is not always acknowledged which of these situations is currently being relevant. These 4 different "use cases" for TL are illustrated in Figure 4.1 and constitute the penultimate column of Table 4.1. In its broadest intent, PL, like TL, is possibly invoked in all of them. However, we only focus on the last one in the restricted scope of this thesis, so it is useful to clearly distinguish them:

(1) Posterior Transfer: In this situation, a source training process has already been done, and has been successful but costly. Now, a new target task has to be tackled, and the user wishes to benefit from this prior experience to improve efficiency on the target. In short, they wish that transfer occurs from the old task to the new task [Taylor and Stone 2009]. See [Tanaka and Yamamura 2003] for an example application to tabular tasks, and [Cornuéjols, Murena, *et al.* 2020] for a recent approach where the transfer function is learned itself.

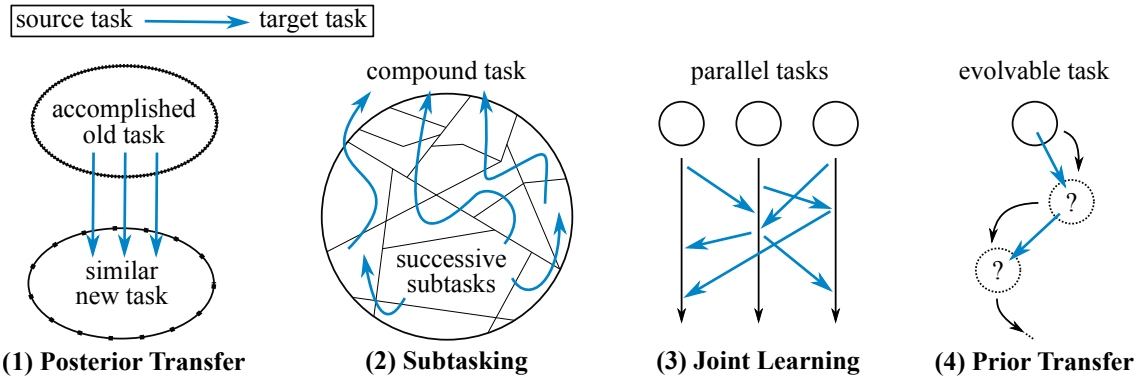


Figure 4.1: Various different learning situations involving Transfer Learning.

(2) Subtasking: In this situation, there is one challenging target task to tackle. Attempts to learning it directly will likely fail, so this target is broken down into several easier, source tasks. The idea is to learn the source tasks first, then successively combine them together to ease the learning path towards the whole target. In short, the user wishes that transfer occurs from the small tasks to the big task [Taylor and Stone 2009]. This relates to the idea of "Curriculum Learning", a developmental approach to learning complicated tasks [Bengio *et al.* 2009; Sigaud and Droniou 2016; Sodhani *et al.* 2020; Narvekar *et al.* 2020; Suro *et al.* 2021]. In "Hierarchical Learning", transfer is also supposed to be eased by modularizing the tasks, see for example [Devin *et al.* 2017] in simulated robots, or [Frans *et al.* 2018] in virtual 3D articulated bodies.

(3) Joint Learning: In this situation, several tasks have to be learned at once. In order to improve efficiency of the overall parallel process, the user expects that any progress made in one task is immediately propagated to the others so the other agents can benefit from it without needing to discover it by themselves. In short, the user wishes that transfer occurs among parallel tasks [Caruana 1994; Teh *et al.* 2017]. See for instance [Harel and Mannor 2011] for an adaptation of this principle to multiple parallel SL tasks called "Multiple Outlooks".

(4) Prior Transfer: In this situation, the task will undergo future changes, but it is unknown yet what these changes will be or when they will occur. Hence, the idea is to design an agent that is flexible enough to adapt these changes, and keep improving on the transforming task by always relying on the accumulated experience. In short, the user wishes that transfer occurs from any task to the next [Ring 1994; Thrun 1995; Ring 1997]. This is closely related to the notions of "Concept Drift", "Continual Learning" and "Lifelong Learning". See for instance [Xu and Zhu 2018] for a use of RL in continual SL, [Seff *et al.* 2017] for an application to Generative Adversarial Networks, or [Rusu *et al.* 2016] for incremental improvement of "Progressive Networks". In RL, the corresponding tasks would belong to the category of "continuing" tasks, because they have no expected termination point.

The PL situation described in Section 1.2, with the roverbot example, is an instance of Prior Transfer **(4)**. Although PL addresses the problem of accommodating signature change in any of the above situations in principle, we consider that agents equipped for Prior Transfer are the most generic ones, because signature changes are unpredictable events in general. In the remainder of this thesis, we focus on PL in the situation **(4)**.

4.2.3 The Challenges of Transfer Learning

TL suffers from a few common pitfalls. First, and like many behavioural search approaches in ML, there are very few theoretical guarantees that the target learning process eventually converges towards acceptable behaviours. In particular, there is no general rule for stating whether any source task is likely to be useful for improving performances on a given target task. The obvious idea is that, the more similar the source and target tasks, the more successful the transfer. But as there is no general, formal description of the dissimilarity between two tasks, the effects of TL remain hard to predict.

Fortunately, in the restricted case of PL as described with the roverbot in Section 1.2, it is expected that the difference between two successive tasks is "slight". Indeed, it only consists in a few elementary signature change events like $(+i,\ -o,\ \sim f,\ etc.)$ occurring on every sensor/actuator update. Apart from this, the environment is supposed to remain the same, and the task is not different from the user perspective. As explained above, there is no formal guarantee that this is "slight" enough for transfer to always occur correctly, but it encourages that TL be used as an inspiration when tackling PL situations.

In addition, one longstanding problem in TL is to correctly "map" the elements of the source task (*e.g.*, inputs, outputs, values, policies, search space) to elements of the target tasks so that the two tasks correctly match [Taylor and Stone 2009]. This is difficult in the general case, but it does fortunately not occur in the PL situation we describe. Indeed, every signature change event like $(+i)$ makes it obvious which input slots are kept from the source task and which one is the original. For this reason again, there are chances that TL be a useful source of inspiration when investigating PL.

This said, there are two major pitfalls in TL that PL also inherits from. First, when an agent succeeds in solving the target task, it possibly happens that it is not able to solve the source task anymore. This phenomenon is known as "Catastrophic Forgetting" (CF).

ANNs tools in particular are prone to CF, and there is a real effort in the community to alleviate this problem. In [Rusu *et al.* 2016] for instance, ANNs are extended with additional parameters on every new task met, so they do not overwrite previous knowledge at the cost of increasing complexity. In [J. Kirkpatrick *et al.* 2017], the learning rate of every weight is finely adjusted depending on its derivative, so the network can keep learning without forgetting, at least until it reaches maximal knowledge capacity. In a similar fashion, Zenke *et al.* 2017 track not only the weights of their networks, but also their past weights and importance, so the new learned policies can be approached by only tuning

weights whose incidence on past knowledge is minimal, at least until maximal capacity is reached. In [Kaplanis *et al.* 2018], bio-inspired modification of the network "synapses" is used to prevent forgetting and enable continual learning.

Although the phenomenon is challenging to alleviate, the various TL situations described in the previous section indicate that CF is only undesirable if the source task is expected to be faced again. To our knowledge, this has not been explicitly stressed in the literature. In curriculum approaches for instance, like Subtasking **(2)**, source tasks *constitute* the target task, so it is obvious that forgetting must be avoided [Sodhani *et al.* 2020]. However, in Posterior Transfer **(1)**, the phenomenon is only undesirable if the target agent is supposed to solve the source task again, either because there is no copy of the source agent left, or because the target agent is expected to eventually solve both tasks, but then the situation is better described as Subtasking **(2)**. In Joint Learning **(3)**, CF is only undesirable if the parallel agents are supposed to be used against any parallel task interchangeably. Otherwise, when each agent is supposed to remain focused on its own task, then it is not a problem that an agent is unable to address the source task it has received transferred knowledge from.

The situation of Prior Transfer **(4)**, is ambiguous in this respect. On the one hand, change events are considered unknown and unpredictable, so it is possible that the agent be occasionally reverted to a situation already encountered in the past [Al-Ghossein *et al.* 2018]. This happens for instance when a roverbot sensor has been broken for a few weeks and then gets fixed. In this case, CF is an undesirable effect of TL, because the bot has to learn how to use the sensor again and then computing resources are wasted. On the other hand, it is sometimes unlikely that any change event be ever reverted. For instance, it is unlikely that a deprecated statistic describing the usage of telegram services be ever used again in predicting customer preferences. Alternately, it is unlikely that the broken roverbot sensor be ever fixed if it happens to be on a mission on Mars. In this situation, not being able to learn again in a situation encountered in the past is not a problem, and there is no need to alleviate CF.

In summary, the transfer situation **(4)** highlights a tradeoff not explicitly mentioned, to our knowledge, in TL literature. The alleviation of CF has a cost, either because it makes the learning algorithm more complex [Rusu *et al.* 2016; Kaplanis *et al.* 2018], or because it consumes a limited agent knowledge capacity [J. Kirkpatrick *et al.* 2017; Zenke *et al.* 2017]. But it is not always necessary, because the source task is not always expected to be faced again by the target agent. In this situation, remembrance of past capacities can be traded for improved performances on the task at hand. In this thesis, as we engage into early investigation of more PL-specific, signature change related problems, we do not attempt to tackle the problem of CF yet. As a consequence, the learning procedures tested in Chapters 6 and 7 are not explicitly designed to alleviate it.

The second major pitfall of TL situations is that the effect of a given transfer procedure is not absolutely reliable. For instance, when a target agent receiving knowledge with a

transfer procedure happens to be better than the naive agent on a target task, it is possible that another one receiving knowledge with the same transfer procedure be less efficient than the naive learner on another target task. This phenomenon is known as "Negative Transfer", and is always undesirable because it cancels the advantages of the TL approach against trivial accommodation of the task change.

Negative transfer can be alleviated under particular circumstances. See for instance [Ge *et al.* 2014] for a study of negative transfer in SL with multiple sources tasks. However, there are very few theoretical guarantees against negative transfer in the general case [Taylor and Stone 2009; Torrey and Shavlik 2010]. In particular, there is no guarantee that PL be not subject to it. For this reason, we specifically designed the experiments presented in Chapters 6 and 7 to address and quantify negative transfer in PL.

Part II

Contributions

Chapter 5

Theory of Protean Learning

This chapter expounds our theoretical contributions to the domain of PL.

The first section is constructed to provide a clear overview of PL, with the various challenges involved and the roadmap toward future progress in the field.

Then, as a groundwork, we develop a rigorous formalization of PL. In current work, formalization is solely used is to make it unambiguous what we refer to as a "signature change", and a "protean" agent in the next. We expect that it lays out the foundation for future analysis of convergence properties of dedicated PL algorithms, which we address with an experimental approach in Chapters 6 and 7 for the sake of extensiveness. The formalization is inspired from the traditional modelling of RL, but explicitly focuses on the streaming nature of OL with formal diagrams, so the dependencies among streams, the retroactions between procedures and the source of the changes become apparent.

After this general modelling has been set up, and for the remainder of this thesis, we restrict our focus to two particular signature change events: $(+i)$ and $(-i)$, constituting the Input-PL (IPL) subdomain. Under the light of these events, we study the structure of the explored landscapes to uncover that they exhibit 10 different possible topological configurations, or *IPL profiles*. We develop a taxonomy of these profiles and study how they possibly influence the reactions of protean agents to the changes.

We also show that there exists a set of natural projections mapping the diminished and augmented search spaces together, so the agent can navigate among them without becoming undefined when $(+i)$ or $(-i)$ occurs. These projections are generic, as they do not depend on the IPL profile at hand or the search space topology. We defend that they constitute good candidates for generic accommodation of signature changes in IPL, and lay out the experimental principles used in the next two chapters to address their value.

5.1 Informal Overview

In the previous chapter, PL was positioned with respect to existing literature. Before we formalize PL, this section makes our overall approach of PL explicit, consistently with the lines at the bottom of Table 4.1. In particular, we sketch a coarse-grained, high-level

overview of PL domain before restricting our focus to the work presented in this thesis. This overview is also a roadmap towards the construction of ideal, full-fledged PL agents.

As written in Chapter 4, signature changes cannot always be predicted. However, they fall into only a few categories that structure the domain of PL:

- The agent possibly benefits from an *addition* of some data slots: inputs $(+i)$, outputs $(+o)$ or feedbacks $(+f)$. For instance, this happens when new sensors or actuators are plugged into the sticky roverbot, or new parallel objectives are defined by the user, like watching the battery level in addition to following the target.

  Note that "addition" possibly refers to different kinds of events. For instance, when something *hidden* has been revealed to the agent, like the battery level, there is a true $(+i)$. On the other hand, when something *new* has started existing, like keyboard input on a new device plugged into the agent, there is both an environmental change $(\sim E)$: the keyboard plug, and *then* an input addition $(+i)$.

- The agent suffers a *removal* of some data slots: inputs $(-i)$, outputs $(-o)$ or feedbacks $(-f)$. For instance, this happens as sensors and actuators break, or as they become obsolete and the user removes them and cancels objectives.

- There is a *change* in the domains of possible values for some data slots: inputs $(\sim i)$, outputs $(\sim o)$ or feedbacks $(\sim f)$. For instance when a temperature sensor widens its sensitivity range, or when a wheel motor is upgraded to also feature backwards spinning, there are two ways to model the situation:

  - In the first modelling option, the set of possible values for the data slots is updated. For instance, if the rover possible output values for the wheel motor are initially ranging in $[0,\ 1]$ with the value 0 meaning stop, the value 0.5 meaning half-speed and the value 1 meaning full speed, then after the $(\sim o)$ event, the new range is $[-1,\ 1]$ and the new values have new meanings like -1 meaning full speed backwards and -0.5 meaning half-speed backwards. Note that the meaning of previously existing values is unchanged. This explicitly states to the agent that the range $[-1,\ 0[$ has not been explored yet, and that new environmental responses are expected whenever these values are tested.
  - In the second modelling option, the set of possible values for the data slots is always the same from the agent perspective, and only the environment accommodates the change. With the same example as above, if the agent possible output values for the wheel motor are initially ranging in $[0,\ 1]$ with the same meanings, then the range after $(\sim o)$ event is still $[0,\ 1]$, but the meanings have changed. For instance, after the change, the value 0 means full speed backwards the value 0.5 means stop, and the value 1 means full speed. This approach is easier to tackle as it reduces any change event $(\sim i)$, $(\sim o)$ or $(\sim f)$ to a simple environmental change $(\sim E)$. As such, it falls out of the scope of PL

and can be tackled with other TL techniques. However, this approach is not equivalent to the former because the agent is given less information about the change, so it is likely that a PL approach yield better results in this situation.

- Beyond elementary operations on data slots, the agent may also *splits* into separate pieces, so the data slots are separated at once into isolated groups. For instance, if the user wishes to extract an independent battery-caring module from the rest of the sticky bot, so as to use it in another artefact, then they have to separate all battery-related sensors, objectives, knowledge *etc.* to build a new agent from them. As this agent has a smaller signature, this higher-level operation falls under the scope of PL.

- Alternately, several agents may *merge* together, so the data slots are combined into a whole. For instance, this happens when the user wishes to import abilities from another static agent able to, say, beep whenever there is a spider nearby. If they expect that the resulting agent takes advantage of its target-following abilities to better track the animals and go beep back close to the user, then the beeping actuators, every spider-related sensors, the new corresponding objectives and the other agent knowledge need to be merged into the sticky bot, in a way similar to traditional curriculum approaches and Subtasking **(2)**. However, the signature of the resulting agent is augmented, so this operation also falls under the scope of PL.

Traditional TL agents are able to face $(\sim E)$ events, in which the signature is fixed. The overall intent of PL is to make online agents also able to face $(+i,\ +o,\ +f,\ -i,\ -o,\ -f,\ \sim i,\ \sim o,\ \sim f)$, but also $(split)$ and $(merge)$ events, without being undefined, and without discarding their previous experience. An ideal PL agent is able to accommodate any of these changes and keeps learning no matter the variations in its signature.

Obviously this is a challenging goal, and the issues involved differ much depending on the change event. For instance, one challenge with $(+i)$ and $(+o)$ is to trigger exploration of possible behaviours again, especially if the agent already satisfyingly converged. This is also the case with $(\sim i)$ and $(\sim o)$ when new possible values are added to the data slots.

With $(-i)$ and $(-o)$, it is expected that performances decrease if the agent was heavily relying on the lost data slots, so the challenge is also to gracefully ease the regression. This is also the case with $(\sim i)$ and $(\sim o)$ when previously possible values become forbidden.

Regarding $(+f)$ and $(-f)$, the multiplication of objectives loosens the definition of the "optimal behaviour" to look for, and it is expected that various behaviours perform equally well on the Pareto front [Yaochu Jin and Sendhoff 2008]. In this situation, one new challenge is for the user to correctly specify their precise expectations. In contrast, a simple $(\sim f)$ event can be considered a simple subset of $(\sim E)$, which is already the object of active research in general TL (see Table 4.1).

The challenges involved also depend much on the learning methods actually applied. For instance, even though ANNs are ubiquitously used as function approximating tools

in ML procedures, they are typically difficult to operate with when it comes to transforming them or extracting knowledge. This makes future PL milestones like *splitting* agents extremely challenging with today's state of the art, but see [Watanabe *et al.* 2018] for an insight into modularizing ANNs.

On the other hand, *merging* ANN-based agents together does not imply that their knowledge be understood or modular, but suggests that a combinatory explosion of possible interactions between all data slots and inner networks states be managed. This constitutes another critical challenge to be met in PL.

In this thesis, we engage a few steps within the domain of PL. First, in Section 5.2, we construct a formalization of this learning situation encompassing the above speculated signature change events. Then, Section 5.3 restricts our focus to only $(+i)$ and $(-i)$ events, constituting the heart of Input-PL (IPL) subdomain. Theoretical considerations on the necessary features of natural IPL transfer projections are developed in this section.

Later, in Chapter 6, we design and conduct an experiment demonstrating the basic viability of IPL in OSL. In particular, with a RNN-based learner, we ensure that the natural projections theorized in Section 5.3 are sufficiently robust to avoid negative transfer in a variety of learning situations.

Finally, Chapter 7 extends the results of Chapter 6 into a tabular RL context (see Figure 2.2). We conclude that the natural projections described in Section 5.3 are also applicable to traditional RL methods like Q-Learning and Actor-Critic. In addition, although the tested projections remain generic, they are sufficient to perform good transfer while accommodating signature changes in a variety of learning situations. In future works, we expect the simplifying hypotheses of tabular contexts to be lifted, and other change events from the wider domain of PL $(\sim i,\ +o,\ -f,\ \textit{etc.})$ to be eventually addressed.

5.2 Formalization

This section develops a rigorous formalization of the PL problem. We start by summarizing the traditional mathematical modelling of RL, before we switch to a data-stream oriented view and extend the situation to PL in general. Finally, we discuss how PL is not only an extension of RL, but a particular case of TL intersecting with any OL situation as illustrated in Figure 2.2.

5.2.1 Background

RL is traditionally formalized using Markov Decision Processes (MDP) [Garcia and Rachelson 2013; Sutton and Barto 2018]. On each time step t, the RL agent perceives a "state" s_t, a random variable taking its values in the space of all possible perceptions $\mathcal{S}$, and a random scalar "reward" r_t in $\mathbb{R}$. The agent is then responsible to pick an "action" a_t in $\mathcal{A}$, according to the probability distribution given by its current internal behaviour

called a "policy" P:

$$a_t \sim P(s_t,\ r_t) \tag{5.1}$$

Note that the distribution is parametrized by the latest inputs s_t and r_t. In reaction to the action, the environment E determines the distribution of the next step state and reward:

$$(s_{t+1},\ r_{t+1}) \sim E(a_t) \tag{5.2}$$

This response distribution is parametrized by the latest action a_t.

To describe how good the agent currently performs, a "discounted" return F_t is defined as the sum of future rewards starting from t:

$$F_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i \tag{5.3}$$

The discount factor $\gamma \in [0,\ 1[$ represents a recency principle. With this factor, rewards in the near future are worth more than distant ones. The factor is applied so that the sum always converges.

With this setting, the behavioural search for a RL agent is expressed as an optimization problem over the space of considered policies $\mathscr{P}$: Find the optimal policy P^* that maximizes the expected return value F at any t:

$$P^* = \arg\max_{P \in \mathscr{P}} \mathbb{E}(F_t | P) \tag{5.4}$$

Time dependency is important here, because the search for P^* is typically performed during the very course of the interaction between the agent and the environment. In other terms, every time a new candidate P is tested, the value of F_t is updated. This is the reason why RL is a case of OL. There exists various common methods to learn in this context. Chapter 7 describes and use two methods known as Q-Learning and Actor-Critic.

5.2.2 PL as a Problem of Stream Processing

In the context of PL, we need to extend the above model to take into account changes in the signature of the agent-environment interface, *i.e.* changes in $\mathscr{S}$ and $\mathscr{A}$. We construct a formal model that focuses on viewing RL as a *stream processing* situation.

Instead of using the concepts of "state", "action" and "rewards", dedicated to RL, we stick to the more general, stream-oriented concepts of *inputs*, *outputs* and *feedbacks*. The exact mapping between these two sets of concepts is loose, as it depends both on perspective and on the learning situation at hand, so it is not straightforward to use the ones as strict replacements to the others. This is developed in Section 5.2.7.

A *data stream* is a value that changes over time. Inputs i (loosely mapped to "states"), outputs o (loosely mapped to "actions") and feedbacks f (loosely mapped to "rewards") of a learning agent are considered data streams. The agent itself is considered a *stream*

processing unit that endlessly transforms inputs i into outputs o with respect to an internal procedure P called its "behaviour" (loosely mapped to "policy"). The objective of RL is that values in the f stream become and remain high.

On the other hand, the environment E is another stream processing unit working the other way round. It endlessly transforms the output streams o into input streams i and feedbacks f, by strict application of the universe rules.

The *signature* describes the interface between the agent and the environment, with the arity and the types of the data streams they are expected to receive and produce. In other words, it is the collection of domains the various streams take their values in. The core idea of this formalization is to consider that the signature is a stream itself, so that it also changes in time and extends RL to PL.

5.2.3 Data Streams and Causality

Streams are represented by functions of continuous time, like $u\colon \mathbb{R}^+ \to \mathscr{D}$. They take their value in arbitrary domains $\mathscr{D}$. Streams time is discretized with arbitrary precision $\varepsilon \in \mathbb{R}^{+*}$ by sequences ${}^{\varepsilon}u\colon \mathbb{N} \to \mathscr{D}$ such that:

$$\forall t \in \mathbb{N},\ {}^{\varepsilon}u(t) = u(\varepsilon t) \tag{5.5}$$

The elementary duration step εt adjusts continuous, real time with any desired accuracy level, while still representing the successive elementary decisional steps of the agent.

As they are processed by the agent or the environment, streams transform into each other. Viewed another way, streams are determined by other streams. We call *determination function* d a function able to determine an outgoing stream v from an incoming stream u no matter the precision ε considered: $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$,

$${}^{\varepsilon}v(t) = d_{\varepsilon}({}^{\varepsilon}u(0),\ \ldots,\ {}^{\varepsilon}u(t-1),\ {}^{\varepsilon}u(t)) \tag{5.6}$$

Stream determination has a *memory*, because current value of v possibly depend on any past values of u. As such, the determination process is not Markovian in general. Alternately, the determination process can be considered Markovian, but then it must be rewritten to involve recurrence and a hidden stream h_f noted h:

$$\begin{cases} {}^{\varepsilon}h(0) = {}^{\varepsilon}d(\varnothing) \\ ({}^{\varepsilon}v(t),\ {}^{\varepsilon}h(t+1)) = {}^{\varepsilon}d({}^{\varepsilon}u(t),\ {}^{\varepsilon}h(t)) \end{cases} \tag{5.7}$$

Note that the initial hidden state $h(0)$ must be defined along with the determination function d.

Stream determination is also *causal* because current and past values of u only determine the current value of v, and not its future values. Further iterations of u are needed to determine future values of v.

The following graphical alias is used to represent determination relation (5.6) or (5.7):

$$u \text{—}(d)\text{→} v \tag{5.8}$$

The symbol in parentheses represents the determination function, the symbol pointed by the arrow head is the consequence stream, and the symbol pointed by the line with no head is the cause stream. For instance, $i \text{—}(P)\text{→} o$ means that the inner agent process P feeds from input stream i to produce the output stream o in a causal, non-Markovian way. In other terms, P is recurrent and exhibits a memory. Conversely, $o \text{—}(E)\text{→} i$ means that the environment works the other way round.

Joining two determination functions in a cycle this way results in a recursive definition for all values $i(t)$ and $o(t)$, so it is ill-defined. To correctly express the agent-environment retroaction loop, we need to bootstrap their dynamics with another type of relation:

$$\begin{cases} {}^{\varepsilon}v(0) = d_{\varepsilon}(\varnothing) \\ {}^{\varepsilon}v(t+1) = d_{\varepsilon}({}^{\varepsilon}u(0),\ \ldots,\ {}^{\varepsilon}u(t)) \end{cases} \tag{5.9}$$

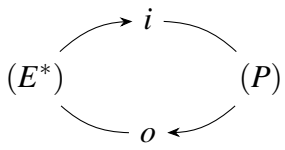
The relation (5.9) introduces a one-step delay determination function. And the determination function is also able to determine the first consequence value $v(0)$ on its own. In the Markovian view involving a hidden state h_d noted h, this translates as:

$$\begin{cases} ({}^{\varepsilon}v(0),\ {}^{\varepsilon}h(0)) = d_{\varepsilon}(\varnothing) \\ ({}^{\varepsilon}v(t+1),\ {}^{\varepsilon}h(t+1)) = d_{\varepsilon}({}^{\varepsilon}u(t),\ {}^{\varepsilon}h(t)) \end{cases} \tag{5.10}$$

This new relation is graphically represented as:

$$u \text{—}(d^*)\text{→} v \tag{5.11}$$

The agent-environment feedback loop can therefore be well-defined with cycling arrows $i \text{—}(P)\text{→} o$ and $o \text{—}(E^*)\text{→} i$, meaning that the environment is responsible for bootstrapping the loop and determines the first inputs to the agent.



(5.12)

This constitutes a rigorous representation of the informal retroaction diagrams used in Chapter 1. In this formal model, P represents the roverbot, E its surrounding environment, i the data streaming from its sensors and o the data flowing to its actuators. (5.12) is equivalent to the following system of equations (5.13). For the sake of readability, all ε

symbols have been dropped, $\forall t \in \mathbb{N}$:

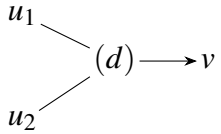
$$\begin{cases} i(0) = E(\varnothing) \\ o(t) = P(i(0),\ \ldots,\ i(t)) \\ i(t+1) = E(o(0),\ \ldots,\ o(t)) \end{cases} \tag{5.13}$$

The above system can also be written as a combination of Markovian processes with hidden states according to interpretations (5.7) and (5.10) of diagram (5.12). This introduces two "hidden" variables h_E and h_P:

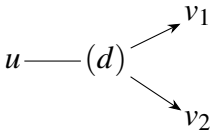
$$\begin{cases} (i(0),\ h_E(0)) = E(\varnothing) \\ h_P(0) = P(\varnothing) \\ (o(t),\ h_P(t+1)) = P(i(t),\ h_P(t)) \\ (i(t+1),\ h_E(t+1)) = E(o(t),\ h_E(t)) \end{cases} \tag{5.14}$$

In the next, we drop the above Markovian interpretation because diagrams and non-Markovian systems are easier to read, but diagrams can always be interpreted this way.

Fleshing out this graphical notation, we use the following aliases when there are multiple cause streams or multiple consequence streams, respectively:



(5.15)



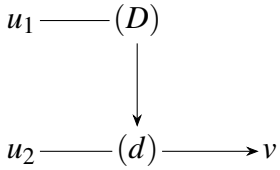
(5.16)

Similarly to eqs. (5.6) and (5.8), the above two diagrams respectively translate as, $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$:

$${}^{\varepsilon}v(t) = {}^{\varepsilon}d\Big(\big({}^{\varepsilon}u_1(0),\ {}^{\varepsilon}u_2(0)\big),\ \ldots,\ \big({}^{\varepsilon}u_1(t),\ {}^{\varepsilon}u_2(t)\big)\Big) \tag{5.17}$$

$$\big({}^{\varepsilon}v_1(t),\ {}^{\varepsilon}v_2(t)\big) = {}^{\varepsilon}d\big({}^{\varepsilon}u(0),\ \ldots,\ {}^{\varepsilon}u(t)\big) \tag{5.18}$$

When the consequence stream values are determination functions themselves, we use the following construct:



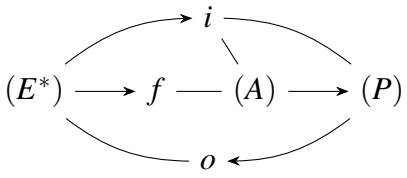
(5.19)

The latter diagram being equivalent to:

$$\begin{cases} {}^{\varepsilon}d(t) = {}^{\varepsilon}D({}^{\varepsilon}u_1(0),\ \ldots,\ {}^{\varepsilon}u_1(t)) \\ {}^{\varepsilon}v(t) = {}^{\varepsilon}d(t)\big({}^{\varepsilon}u_2(0),\ \ldots,\ {}^{\varepsilon}u_2(t)\big) \end{cases} \tag{5.20}$$

Note that ${}^{\varepsilon}d(t)$ is a function in this case.

Equipped with these formal diagrams, we can reformulate the dynamical relation between a RL agent and its environment in a data-stream oriented fashion:



(5.21)

The above representation includes the stream of rewards f produced by the environment, and the agent internal procedure A. A performs the inner search S for good candidate behaviours P, guided by f. Note that A also feeds on i. This is useful for instance to construct internal representations of E, like the value function in QL methods (see Section 2.2.3).

In contrast, A cannot feed on o for obvious causality reasons. This seems to contradict with numerous RL methods, which do take o values into account in practice to improve their policy search. As a matter of fact, no value $o(t)$ can be used before being actually produced, but the methods are still correctly represented without changing the diagram: either consider that $o(t)$ values are determined by A then embedded within $P(t)$, so A memorizes them; Or consider that every $o(t)$ is re-emitted by E within $i(t+1)$, and ignored by P, so A accesses them on subsequent steps.

This representation also makes it clear why RL is a case of OSL: A needs to wait for the next input value $i(t+1)$ to produce the next policy $P(t+1)$, so the two timelines described in Section 2.2.4.1 (the input timeline and the search timeline) line up.

Now that RL is expressed in the above representation, we need to extend it so as to encompass signature changes and the PL situation. In the next section, we construct one particular type of data streams to represents signature change events.

5.2.4 Multiple Streams and Signatures

Signature changes imply that the arity and type of the data streams processed by the agent vary. To represent this, we define one particular type of streams called *multiple streams*.

A multiple stream u, also noted $u^{\Delta,\chi}$, carries both a stream of *domains* noted u^{Δ}, whose values are called *signatures*, and a stream of *values* noted u^{χ} (Fig. 6.1). A signature is a tuple of domains $(\Delta_1, \Delta_2, \ldots)$ and values are elements from these domains $(\chi_1 \in \Delta_1,\ \chi_2 \in \Delta_2,\ \ldots)$. For instance, at $t = 0.9$, the sticky roverbot described in Chapter 1 is sensitive to both "direction to user" and "ground speed", and receives the signature and values:

$$u(t) = u^{\Delta,\chi}(t) = \begin{pmatrix} u^{\Delta}(t) \\ u^{\chi}(t) \end{pmatrix} = \begin{pmatrix} [0,\ 2\pi] & \mathbb{R}^{+} \\ 0.2 \text{ rad} & 15 \text{ cm.s}^{-1} \end{pmatrix} \tag{5.22}$$

The particularity of PL, in contrast with RL, is that the signature stream is not constant. We call *signature change* of the PL agent any variation of u^{Δ} resulting in that the agent later receives values with different domain signatures, thus the term *protean*. For instance,

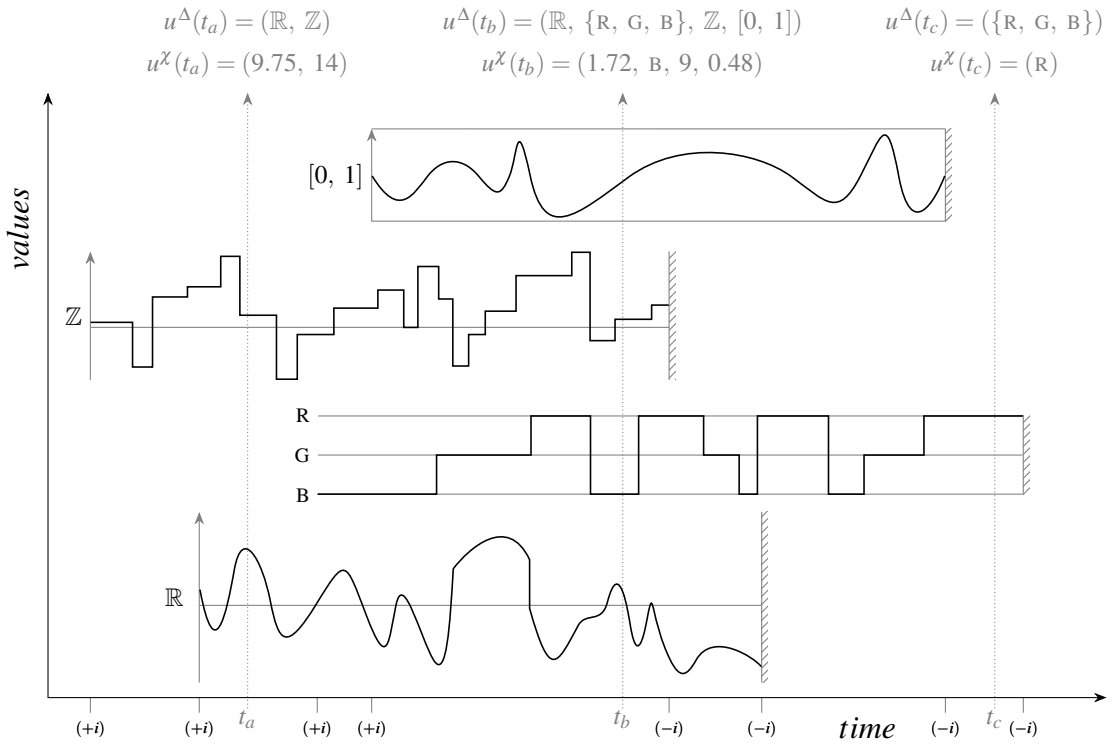


Figure 5.1: Example multiple stream u, also noted $u^{\Delta,\chi}$. Each curve corresponds to one transient domain in the signature stream u^Δ. If u represents an agent input stream, then this learner is initially sensitive to a data valued in $\mathbb{Z}$. The learner later becomes also sensitive to data valued in $\mathbb{R}$, then in $[0,\ 1]$. All these input slots are eventually lost between t_b and t_c. In the end, the learner is only sensitive to a nominal parameter in $\{\text{R},\ \text{G},\ \text{B}\}$. All beginning and end of sensitivities are marked as input addition $(+i)$ or input deletion $(-i)$ events.

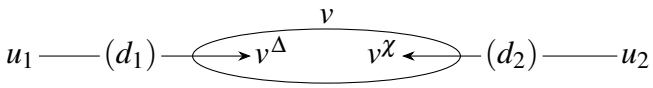
at $t + \delta t = 1.1$, after its front camera has been broken $(-i)$, and a new battery sensor has been plugged in $(+i)$, the sticky agent receives "ground speed" and "battery level" as:

$$u(t+\delta t) = u^{\Delta,\chi}(t+\delta t) = \begin{pmatrix} u^\Delta(t+\delta t) \\ u^\chi(t+\delta t) \end{pmatrix} = \begin{pmatrix} \mathbb{R}^+ & \{1,\ \ldots,\ 5\} \\ 31\ \text{cm.s}^{-1} & \text{level 4} \end{pmatrix} \tag{5.23}$$

Multiple stream possibly make successive, overlapping "tracks" appear on the stream input profile as represented in Figure 6.1. These tracks relate to the global agent signature, so they are not mandatorily processed by distinct submodules within the agent. In the framework called Horde for instance, a set of Generalized Value Functions (GVFs) is used as a modularized approach to learn more from a reinforcement environment than only a value function [Sutton, Modayil, *et al.* 2011]. Although it is an option that every GVF only focuses on one input track while ignoring the other ones, change events affects the arity of all GVFs simultaneously in general.

It is possible that the signature stream u^Δ and the values stream u^χ are caused by two different determination functions. In this situation, we use the following construct with a

simple closed curve to distinguish between u^Δ, u^χ and the combined stream u:



(5.24)

Consistently with eqs. (5.6) and (5.8), the above is an alias for, $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$:

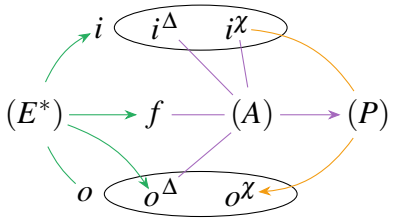
$$\begin{cases} {}^\varepsilon v^\Delta(t) = {}^\varepsilon d_1\big({}^\varepsilon u_1(0),\ \ldots,\ {}^\varepsilon u_1(t)\big) \\ {}^\varepsilon v^\chi(t) = {}^\varepsilon d_2\big({}^\varepsilon u_2(0),\ \ldots,\ {}^\varepsilon u_2(t)\big) \end{cases} \tag{5.25}$$

Note that diagram (5.24) and eq. (5.25) are only consistent with the definition of a multiple stream if every value in $v^\chi(t)$ belongs to the corresponding domain in $v^\Delta(t)$ for every t.

In summary, multiple streams enable seamless representation of data streams with variable type and arity, like the one represented in Fig. 6.1. In the next section, we use them to specify the PL situation with a formal diagram.

5.2.5 Learning Dynamics

At the highest level, a PL situation is represented by 3 multiple streams $(i,\ o,\ f)$, one stream of determining functions P and two fixed determining functions $(E,\ A)$, with the following determination diagram:



(5.26)

According to the interpretation rules defined above, the diagram (5.26) is equivalent to the set of formal equations (5.27–5.30). They form a dynamical system for any time precision $\varepsilon \in \mathbb{R}^{+*}$. For the sake of readability, all ε symbols have been dropped:

$$\begin{cases} \Big(i(0),\ f(0),\ o^\Delta(0)\Big) = E(\varnothing) & (5.27) \\ P(t) = A\Big(\Big(i(0),\ f(0),\ o^\Delta(0)\Big),\ \ldots,\ \Big(i(t),\ f(t),\ o^\Delta(t)\Big)\Big) & (5.28) \\ o^\chi(t) = P(t)\Big(i^\chi(0),\ \ldots,\ i^\chi(t)\Big) & (5.29) \\ \Big(i(t+1),\ f(t+1),\ o^\Delta(t+1)\Big) = E\Big(o(0),\ \ldots,\ o(t)\Big) & (5.30) \end{cases}$$

Note that $P(t)$ is a determination function itself. Each colored equal sign corresponds to one determination arrow in (5.26).

There are two major differences with the traditional RL retroaction drawn in (5.21). First, all basic data streams i, o and f in (5.26) are multiple streams now, so their signature

can change any time. Second, while output values o^{χ} are still determined by P, output signatures o^{Δ} are determined by E. In other terms, the agent is not responsible for its current input/output signature. Instead, A is responsible for always producing P candidates with adequate signature, based on the information it gets from i^{Δ} and o^{Δ}.

In the end, only two objects are not depending on time in this system: the environment E and the inner agent strategy A. In the sense of dynamical systems, E and A embody the evolution *rules* of the system, while its *initial state* is represented as the first production of E: $\big(i(0),\ f(0),\ o^{\Delta}(0)\big)$.

5.2.6 The Objective of PL

Similarly to RL, PL objective is to optimize values in the feedback stream f [Sutton and Barto 2018], but the protean nature of f make this less straightforward to formulate.

First, not every domain is suitable for the signature of the reward stream f^{Δ}, because feedback values need to be *compared* to one another so as to decide which one is "best". For instance, scalar values are useful, like values in $\mathbb{R}$. But categorical values like $\{\text{R},\ \text{G},\ \text{B}\}$ (see Figure 6.1), which can be used as regular inputs i, cannot be used as feedback values unless at least partially ordered, as discussed in Section 2.1.1.

Second, the reward stream f is a multiple stream, so there exists, in the general case, a wide range of non-Pareto-dominant "optimal" behaviours to search. In addition, even marginal optimality is not well defined with respect to only one discounted temporal stream of feedbacks like in eq. (7.61), because PL data streams cannot be assumed to last forever. As a consequence, the traditional learning objective of "maximizing rewards" needs to be reformulated in PL as a multi-objective ML goal:

Given an environment E in the dynamical system (5.26), with a multiple stream of feedbacks f whose signature f^{Δ} only contains partially ordered domains, find an agent procedure A such that all values taken by the values stream f^{χ} are Pareto-optimized.

An agent fulfilling the above requirement ideally solves the task represented by E. However, the longstanding quest of ML, and of AI in general, is not only to construct an ideal agent able to solve one particular task, but to construct an ideal *generic* agent, supposed to seamlessly address a wide range of different tasks. Consequently, and in addition to the "small" PL objective defined above, the "big" objective of PL is better described as:

Given a class of environments $\mathscr{E}$ whose every element E fits into the dynamical system (5.26), with multiple streams of feedbacks f whose signatures f^{Δ} only contain partially ordered domains, find an agent procedure A such that all values taken by the values stream f^{χ} in every $E \in \mathscr{E}$ are Pareto-optimized.

An agent fulfilling the above requirement adapts to various changing environments no matter the variable nature of the signatures streams i^{Δ}, o^{Δ} or f^{Δ}. The wider $\mathscr{E}$ in terms of the represented range of tasks, the more flexible the agent.

5.2.7 Discussion

The formalism presented here is novel, and offered as a contribution of this thesis to better understand the scope of PL in our current vision, previously described in Chapter 3. For instance, our model states that all multiple streams domains are emitted by E, which rules out the possibility for an agent to decide when the signature changes happen. As such, a learning agent able to upgrade its own sensors/actuators capabilities at will falls outside the strict scope of PL, unless the upgrade is an indirect environmental result of its actions[1].

For the sake of simplicity, our representation of PL does not yet feature the traditional probabilistic view of RL [Sutton and Barto 2018]. Instead, it focuses on the agent *signature* and how it changes in time, which is essential to PL. This said, we believe it is easily extended to stochastic processes, by replacing the deterministic interpretations of the diagrams by probabilistic relations. For instance, instead of being interpreted as an equation, the following diagram:

$$u \text{ —} (f) \rightarrow v \tag{5.31}$$

is interpreted as a probabilistic connection between two random processes u and v, with f a probability distribution parametrized by past realizations of u, *e.g.*, $\forall \varepsilon \in \mathbb{R}^{+*},\ \forall t \in \mathbb{N}$

$${}^{\varepsilon}v(t) \sim {}^{\varepsilon}f({}^{\varepsilon}u(0),\ \ldots {}^{\varepsilon}u(t)) \tag{5.32}$$

In summary, and consistently with Box 1.1, we distinguish at least 3 different kinds of learning agent policies $i \text{ —} (P) \rightarrow o$ (with dropped ${}^{\varepsilon}$, and ${}^{\chi}$ symbols):

- *Deterministic* policies: The current input $i(t)$ predicts output $o(t)$ exactly, like in:

$$o(t) = P\big(i(t)\big) \tag{5.33}$$

  This is the most basic type of policy, easily represented as a plain function of inputs.

- Non-recurrent *stochastic* policies: The outputs are randomly determined depending on current input values, like in

$$o(t) \sim P\big(i(t)\big) \tag{5.34}$$

  In this case, P is better represented as a probability distribution parametrized by inputs. The agent is less predictable, but Chapter 7 shows that it possibly lead to better performances against non-deterministic environments.

[1]Note that, similarly to SL and UL being useful as technical elements of RL, PL is also useful as a methodological element of other learning situations like this one. An agent capable of accommodating unforeseen signature changes is likely advantaged when able to upgrade signature on its own.

- *Recurrent* policies: The agent has a *memory* of past inputs, and uses it to determine the current output, like in:

$$o(t) \simeq P\big(i(t),\ i(t-1),\ \ldots,\ i(0)\big) \tag{5.35}$$

Where the symbol $\simeq$ is used to represents either $=$ or $\sim$. Equivalently, and according to (5.7), the agent contains a *hidden state h* recurrently updated:

$$\begin{cases} h(0) \simeq P(\varnothing) \\ \big(o(t),\ h(t+1)\big) \simeq P\big(i(t),\ h(t)\big) \end{cases} \tag{5.36}$$

This is the most sophisticated type of policies because it makes full use of the possibilities offered by the $i \multimap (P) \rightarrowtail o$ model.

Agent capabilities and reactions to PL differ depending on the type of policies explored by A. Chapter 6 addresses PL in a simple learning situation with only deterministic policies. Chapter 7 addresses PL with both deterministic and non-recurrent stochastic policies. Recurrent policies will be addressed in future works.

This section has introduced PL under the form of an extension to RL, but the connection between PL and other fields of ML is more subtle, (see Figure 2.2). This is where the mapping between the concepts of *input* and "state", of *output* and "action" *etc.* is refined.

For instance, PL can be considered a *special case* of RL. Under this perspective, not only the inputs are given as RL "states", but also their signature, *e.g.*, $s_t \in \mathcal{S}$ with:

$$s_t = \big(i^\Delta(t),\ i^\chi(t)\big). \tag{5.37}$$

Alternately, PL can also be considered a *generalization* of RL. Under this perspective, the signature is no longer considered constant, *e.g.*, $s_t \in \mathcal{S}_t$ with variable $\mathcal{S}$, for instance:

$$\mathcal{S}_t = \prod i^\Delta(t). \tag{5.38}$$

This formalism is also compatible with future extensions of PL where the cause for signature changes is that PL agents compose each other. For instance when one agent splits into two or when two agents A_1 and A_2 merge into one agent A, their inputs are separated or joined with *e.g.*: $i^\Delta(t+\delta t) = i_1^\Delta(t) \cup i_2^\Delta(t)$.

As discussed in Section 4.2, PL is also an instance of TL. Whenever there is a change in the streams domains i^Δ, o^Δ or f^Δ, the agent A is facing a transfer problem because the task at hand has evolved. A new candidate P must be found to accommodate the change, and previous knowledge needs to be reused so as to ensure better performances than a naive agent. The model of PL offered in diagram (5.26) therefore fits into the frame of TL, Concept Drift and Continual Learning.

In addition, as a generic instance of Prior Transfer **(4)**, the model of PL offered in diagram (5.26) is also suited to represent OL tasks in general, including RL but also OSL. To interpret (5.26) in the context of OSL, consider the following:

- The data streams i and o carry the successive learning batches.
- The environment E holds the training set $\mathcal{T}$ containing $(i^\star,\ o^\star)$ pairs.
- Every value $o^\chi(t)$ produced by the agent A is an attempt to mimic the received input pair $i^\chi(t) = (i^\star(t),\ o^\star(t))$.
- The environment E compares the stream o^χ to $o^\star$, computes the corresponding *loss* stream f and feeds it back to A.

The major difference with RL is that $i(t)$ never depends on past values of o. Also, $f(t)$ is always immediately available after $o(t)$ has been emitted. In other words, in OSL, E is always Markovian with no hidden state, so A has no credit assignment problem to solve. This is the reason why OSL is easier than RL in general. The experiment presented in Chapter 6 assesses basic viability of a low-level PL projection technique in an OSL context, and the experiment presented in Chapter 7 addresses this in a RL context.

5.3 Input Addition and Deletion

In the remainder of this thesis, we focus on two specific signature change events in PL: input addition $(+i)$ and input deletion $(-i)$. This restricts PL to Input-PL (IPL). This section studies how these events affect the search space $\mathscr{P}$ explored during learning. We see how the IPL search landscapes fall into only a few categories of typical *profiles* illustrated in Figure 5.2. We also see how the *transfer* component of IPL essentially relates to the choice of insightful *projections* between "diminished" and "augmented" search spaces.

5.3.1 Time Dependency

During learning, the agent explores the landscape of $\mathscr{P}$ with respect to an objective function F representing the IPL learning objective. This is a maximization problem, and the regions of $\mathscr{P}$ that yield highest values of F need to be found. Every element $P \in \mathscr{P}$ is a policy able to process inputs into outputs according to $i^\chi \multimap (P) \rightarrow o^\chi$. As a requirement, P processes elements of the input space $\mathscr{I}$ into elements of the output space $\mathscr{O}$, with:

$$\mathscr{I} = \prod_{\Delta \in i^\Delta} \Delta \tag{5.39}$$

$$\mathscr{O} = \prod_{\Delta \in o^\Delta} \Delta \tag{5.40}$$

The above two Cartesian products constitute the signature of P. For the policy search to be well-defined, every procedure in $\mathscr{P}$ must have this same signature, or is ill-defined and cannot be tested as a candidate.

When $(+i)$ and $(-i)$ events come into play, i^Δ varies in time, and so does $\mathscr{I}$. For instance, when only a $(+i)$ event occurs between times t_1 and t_2 with $t_1 < t_2$, then:

$$\mathscr{I}(t_2) = \mathscr{I}(t_1) \times \Delta^+ \tag{5.41}$$

Where Δ^+ is the domain added to the signature. In this situation, we refer to $\mathscr{I}(t_2)$ as the *augmented* input space, and $\mathscr{I}(t_1)$ is the *diminished* one. Since the augmented signature differs from the diminished one, then no element in the search space can be candidate anymore. As a consequence, $\mathscr{P}$ also needs to vary in time, and upgrade from $\mathscr{P}(t_1)$ to $\mathscr{P}(t_2)$ such that every augmented candidate within $\mathscr{P}(t_2)$ has a signature compatible with $\mathscr{I}(t_2)$.

Conversely, when only a $(-i)$ occurs between t_1 and t_2, then one domain Δ^- is withdrawn from the input signature Cartesian product. In this situation, $\mathscr{I}(t_1)$ is the augmented space and $\mathscr{I}(t_2)$ is the diminished space. No candidate in $\mathscr{P}(t_1)$ can produce output in response to $i^\chi(t_2)$ values anymore because there are inputs missing, so the search space needs to be downgraded to $\mathscr{P}(t_2)$.

To abstract over $(+i)$, $(-i)$, t_1 and t_2 in the next, we refer to the *augmented* input space and search space with the regular symbols $\mathscr{I}$ and $P \in \mathscr{P}$, while we refer to the *diminished* input space and search space with the barred symbols $\bar{\mathscr{I}}$ and $\bar{P} \in \bar{\mathscr{P}}$.

5.3.2 Diminished *vs.* Augmented Search Spaces

Both input spaces $\bar{\mathscr{I}}$ and $\mathscr{I}$ are Cartesian products of domains. The only difference is that $\bar{\mathscr{I}}$ is missing one term in the product compared to $\mathscr{I}$. As a consequence, there exists a natural surjection of $\mathscr{I}$ into $\bar{\mathscr{I}}$, noted with double headed arrow $\twoheadrightarrow$, where every $i \in \mathscr{I}$ maps to the $\bar{i} \in \bar{\mathscr{I}}$ with same values on all dimensions except the missing one:

$$\mathscr{I} \twoheadrightarrow \bar{\mathscr{I}} \tag{5.42}$$

Therefore, for every diminished procedure $\bar{P}$, there exists one trivial augmented procedure P that "mimics" $\bar{P}$ by discarding the additional received value. To construct this P, receive augmented input i, transform into $\bar{i}$ using the above natural mapping, then apply $\bar{P}$. To illustrate this with the roverbot example, consider that a new battery sensor is plugged into the agent, but the robot behaviour remains the same because it just ignores it. This construction is always possible unless the agent has no way to "ignore" the new data.

From now on, we assume that every augmented search space contains such trivial degenerated procedures, so there exists a natural injection of $\bar{\mathscr{P}}$ into $\mathscr{P}$, noted with tailed arrow $\rightarrowtail$, that we refer to as ℓ:

$$\ell : \bar{\mathscr{P}} \rightarrowtail \mathscr{P} \tag{5.43}$$

This has interesting consequences regarding the landscape of the agent objective function F. Namely, landscapes only fall into 10 possible categories that we refer to as landscape *profiles*, illustrated in Section 5.3.4. These profiles provide key insights into the possible reactions of a learning agent to $(+i)$ and $(-i)$ signature change events.

In the next, we describe necessary relations between various subsets of $\bar{\mathscr{P}}$ and $\mathscr{P}$. After listing these relations, we interpret them under the light of behavioural search to expound these various profiles and how they must influence learning under IPL conditions.

5.3.3 Structure of the Search Spaces

To begin with, consider that there is always at least one possible diminished policy, otherwise there would be no point in searching $\bar{\mathscr{P}}$:

$$\bar{\mathscr{P}} \neq \varnothing \tag{5.44}$$

Also, consider that the injection (5.43) is *strict*, otherwise there would be no point to the augmented search space. This implies that there exists a non-empty set $\check{\mathscr{P}}$ containing every non-degenerated augmented policy:

$$\check{\mathscr{P}} = \mathscr{P} - \ell\left(\bar{\mathscr{P}}\right) \tag{5.45}$$

$$\check{\mathscr{P}} \neq \varnothing \tag{5.46}$$

Note that $\check{\mathscr{P}} \subsetneq \mathscr{P}$, so the identity constitutes a trivial, strict injection between both sets:

$$\check{\mathscr{P}} \rightarrowtail \mathscr{P} \tag{5.47}$$

The situation is summarized by the following diagram:

$$\begin{array}{ll} \textit{(augmented policies)} & \check{\mathscr{P}} \rightarrowtail \mathscr{P} \\ \textit{(diminished policies)} & \bar{\mathscr{P}} \quad \nearrow_{\ell} \end{array} \tag{5.48}$$

Note that $\ell\left(\bar{\mathscr{P}}\right)$ and $\check{\mathscr{P}}$ form a partition of $\mathscr{P}$. This diagram is progressively fleshed out in the next, and in Chapter 7.

The learning agent objective is to find a policy that maximizes the function F. As such, consider the optimal diminished policy set $\bar{\mathscr{P}}^*$. This non-empty set contains the best policies that a diminished agent possibly finds in $\bar{\mathscr{P}}$. The associated best policy value is $\bar{F}^*$, with:

$$\begin{aligned} \bar{F}^* &= \max_{\bar{P} \in \bar{\mathscr{P}}} (F(\bar{P})) \\ \bar{\mathscr{P}}^* &= \left\{ \bar{P}^* \in \bar{\mathscr{P}},\, F(\bar{P}^*) = \bar{F}^* \right\} \\ \bar{\mathscr{P}}^* &\neq \varnothing \end{aligned} \tag{5.49}$$

Note that $\bar{\mathscr{P}}^* \subset \bar{\mathscr{P}}$, and that possibly $\bar{\mathscr{P}}^* = \bar{\mathscr{P}}$. As a consequence, the relation between both sets is non-strict. This is important in the next.

On the other hand, $\check{\mathscr{P}}^*$ contains the best policies that a diminished agent cannot possibly find. The associated best policy value is F^*, with:

$$F^* = \max_{P \in \mathscr{P}}(F(P)) \tag{5.50}$$

$$\mathscr{P}^* = \{P^* \in \mathscr{P},\ F(P^*) = F^*\} \tag{5.51}$$

$$\check{\mathscr{P}}^* = \mathscr{P}^* \cap \check{\mathscr{P}} \tag{5.52}$$

Similarly, there is $\check{\mathscr{P}}^* \subset \check{\mathscr{P}}$ and the relation is non-strict. This is summarized in the following diagram:

$$\begin{array}{ll} \textit{(augmented policies)} & \check{\mathscr{P}}^* \subseteq \check{\mathscr{P}} \rightarrowtail \mathscr{P} \\ & \qquad\quad \nearrow_{\ell} \\ \textit{(diminished policies)} & \bar{\mathscr{P}}^* \subseteq \bar{\mathscr{P}} \end{array} \tag{5.53}$$

$\bar{\mathscr{P}}^*$ is non-empty, but possibly $\check{\mathscr{P}}^* = \varnothing$. This happens when the only optimal augmented policies are degenerated. This trivial relation constitute another non-strict relation:

$$\varnothing \subseteq \check{\mathscr{P}}^* \tag{5.54}$$

To avoid cluttering, we highlight the $\check{\mathscr{P}}^*$ symbol with a dotted underline in the diagram, as a reminder of the non-strict relation (5.54):

$$\begin{array}{ll} \textit{(augmented policies)} & \underset{\cdots\cdots}{\check{\mathscr{P}}^*} \subseteq \check{\mathscr{P}} \rightarrowtail \mathscr{P} \\ & \qquad\quad \nearrow_{\ell} \\ \textit{(diminished policies)} & \bar{\mathscr{P}}^* \subseteq \bar{\mathscr{P}} \end{array} \tag{5.55}$$

A diminished agent following a diminished policy $\bar{P}$ has the same end behaviour as an augmented agent following the corresponding degenerated policy $\ell(\bar{P})$, so we assume that both policies have the same value:

$$F(\bar{P}) = F(\ell(\bar{P})) \tag{5.56}$$

As a consequence, since $\bar{\mathscr{P}} \rightarrowtail \mathscr{P}$ the value of best diminished policies cannot be higher than the value of best augmented policies:

$$\bar{F}^* \leqslant F^* \tag{5.57}$$

In other terms, diminished agents cannot perform better than augmented agents, because augmented agents have a larger panel of possible behaviours. Note that the above relation is also non-strict, so that either $\bar{F}^* < F^*$ (the augmented agent does better) or $\bar{F}^* = F^*$ (the diminished agent does as good).

In the end, the above key relations between policies sets and the associated optimal values are all summarized in the following diagram. The symbol $\triangleright$ is used to connect

optimal values to the set they maximize:

$$\begin{array}{ll} \textit{(augmented policies)} & F^* \triangleright \check{\mathscr{P}}^* \subseteq \check{\mathscr{P}} \rightarrowtail \mathscr{P} \\ & \wr\!\!\vee \qquad\qquad \nearrow \ell \\ \textit{(diminished policies)} & \bar{F}^* \triangleright \bar{\mathscr{P}}^* \subseteq \bar{\mathscr{P}} \end{array} \tag{5.58}$$

These relations constitute the structure of the policy search landscape in IPL, and of the interplay between $\bar{\mathscr{P}}$ and $\mathscr{P}$.

5.3.4 The IPL Landscape Profiles

The diagram (5.58) contains a total of four non-strict relations (including relation (5.54)). As a consequence, PL search landscapes exhibit various possible structures, depending on whether the $\subseteq$ relations are strict ($\subsetneq$) or equal ($=$), and whether the $\leqslant$ relation is strict ($<$) or equal ($=$).

Not every combination is possible. Obviously for instance:

$$\check{\mathscr{P}}^* = \varnothing \implies \check{\mathscr{P}}^* \subsetneq \check{\mathscr{P}} \tag{5.59}$$

Also, an empty $\check{\mathscr{P}}^*$ implies that every augmented optimal policy is degenerated, so the augmented agent cannot perform better than the diminished one in this case:

$$\bar{\mathscr{P}}^* = \varnothing \implies \bar{F}^* = F^* \tag{5.60}$$

This leaves only 10 possible combinations, corresponding to the landscape profiles illustrated in Figure 5.2. The various profiles are described by three properties, corresponding to the state of the non-strict relations involving $\bar{\mathscr{P}}^*$, $\check{\mathscr{P}}^*$, and the tuple $(\bar{F}^*,\ F^*)$:

- Regarding $(\bar{F}^*,\ F^*)$, there are two possible options:

  FT: $\bar{F}^* = F^*$: No augmented policy does better than the best diminished policies. We refer to these landscapes as *flat through* the change event or "FT". In FT environments (left column in Figure 5.2), the best agent performance cannot be improved after a $(+i)$ event, because no better performance is possibly achieved with augmented policies. Conversely, removing an input with $(-i)$ is not dramatic for the agent because its theoretical performances do not decrease, only it has to search $\bar{\mathscr{P}}$ again to restore them.

  HT: $\bar{F}^* < F^*$: At least one augmented policy does better than the best diminished policies. We refer to these landscapes as *hilly through* the change event or "HT". In HT environments (right column in the figure), $(-i)$ events are guaranteed to decrease maximal performances. Conversely, there are chances that providing new inputs to the agent with $(+i)$ eventually helps in improving on the task at hand.

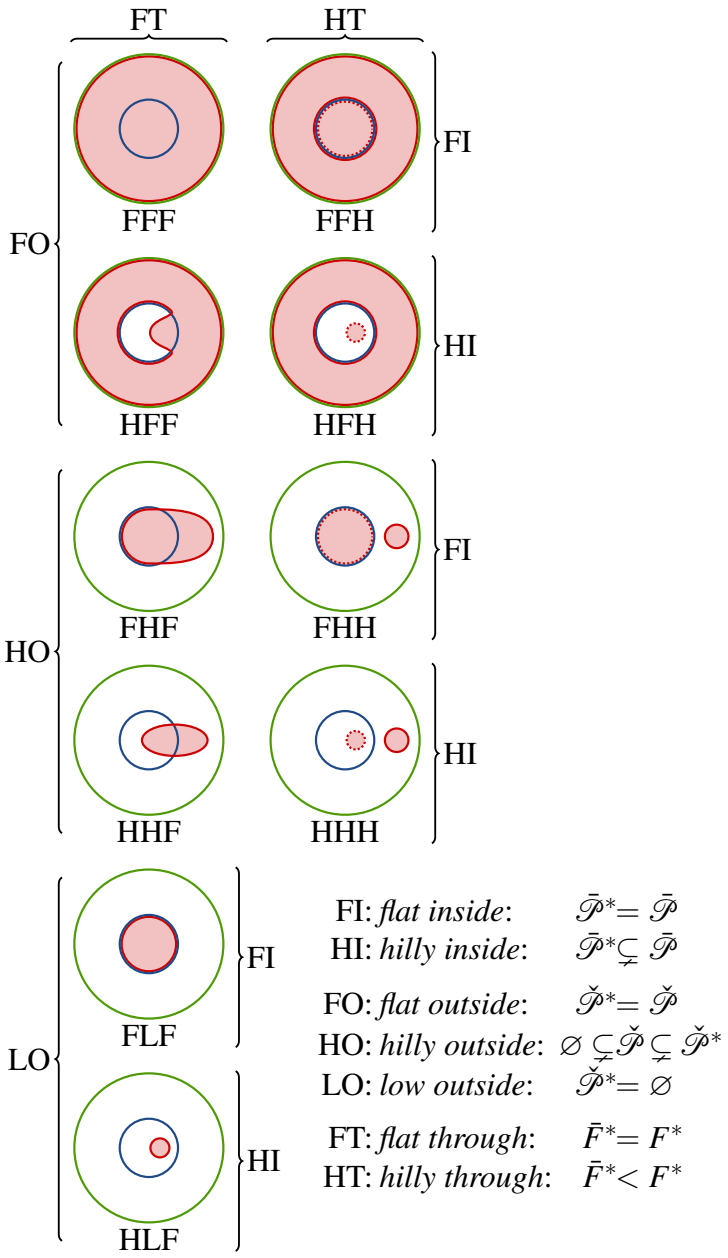


Figure 5.2: Euler diagrams for the various possible environment profiles with respect to IPL events $(+i)$ and $(-i)$. $\mathscr{P}$: augmented policies (outer green disk). $\bar{\mathscr{P}}$: diminished policies (inner blue disk represents $\ell(\bar{\mathscr{P}})$). $\check{\mathscr{P}}$: non-degenerated augmented policies (outer ring). $\mathscr{P}^*$: optimal augmented policies (red area with solid outline). $\bar{\mathscr{P}}^*$: optimal diminished policies (red area within inner disk represents optimal degenerated augmented policies, outline is dotted when not included in $\mathscr{P}^*$). $\check{\mathscr{P}}^*$: optimal non-degenerated policies (red area within outer ring).

- Regarding $\bar{\mathscr{P}}^*$, there are two possible options:

  FI: $\bar{\mathscr{P}}^* = \bar{\mathscr{P}}$: Every diminished policy is optimal, so the search landscape is *flat inside* the diminished policies set, or "FI". In FI environments (odd lines in the figure), removing an input with $(-i)$ also removes the learning challenge, because no decision taken by the agent alters its performance from now on.

  HI: $\bar{\mathscr{P}}^* \subsetneq \bar{\mathscr{P}}$: Some diminished policies are better than others, so the search landscape is *hilly inside* the diminished policies set, or "HI". In HI environments (even lines in the figure), the agent search is not trivial even after a $(-i)$ event, for there are still both policies to prefer and policies to avoid.

- Regarding $\check{\mathscr{P}}^*$, there are three possible options:

  FO: $\check{\mathscr{P}}^* = \check{\mathscr{P}}$: Every augmented policy is optimal provided it is not degenerated, so the search landscape is *flat outside* the diminished policies set, or "FO". In FO environments (upper block in the figure), the augmented agent performs best as long as it does not ignore its additional input channel, which likely makes the augmented search space easy to search.

  HO: $\varnothing \subsetneq \check{\mathscr{P}}^* \subsetneq \check{\mathscr{P}}$: Not every non-degenerated augmented policy is optimal, so the search landscape is *hilly outside* the diminished policy set, or "HO". In HO environments (middle block in the figure), the augmented agent search is less trivial because the optimal policies need to be discovered among suboptimal non-degenerated augmented policies.

  LO: $\check{\mathscr{P}}^* = \varnothing$: Every optimal augmented policy is degenerated, so the search landscape is *low outside* the diminished policies set, or "LO". In LO environments (lower block in the figure), the additional input is never needed to attain the best possible behaviour. As a consequence, $(+i)$ events only bring deceptive information to the agent, since they augment the size of the search space without enabling better behaviours. Conversely, once an augmented agent has converged towards a best policy P^*, it is likely that $(-i)$ event be trivial to accommodate, because there is a trivial new best policy $\bar{P}^* = \ell^{-1}(P^*)$.

These landscape profiles describe the overall structure of IPL search landscape, and the various possible reactions of a learning agent to $(+i)$ or $(-i)$ events. For the sake of brevity, they are referred to with three letters corresponding in order to their *inside* status (F-lat or H-illy), their *outside* status (F-lat, H-illy or L-ow), and their *through* status (F-lat or H-illy). For instance, HLF refers to the *hilly inside*, *low outside* and *flat through* landscape (see Figure 5.2).

5.3.5 Using IPL Profiles

Although they are illustrative, the landscape profiles only contain very general information so they only scratch the surface of the various IPL situations. For instance, it is correct

to assume that the behavioural search is always "trivial" or "easy" when the search landscape is flat (*e.g.*, in the FFF condition), but it is difficult to address how "difficult" is the search within hilly landscapes (*e.g.*, in the HLF condition).

In the next, we summarize several important sources of variability regarding the possible learning trajectories of agents undergoing IPL events, which are not captured by the above profiles. When a profile is skewed, we can express the phenomenon by comparing it to other profiles. We conclude that the profiles are useful to picture and characterize the variety of concrete situations that IPL agents are confronted to.

First, the non strict relations in diagram (5.58) are not quantified. They only distinguish between generic and degenerated cases. The condition FFF (FT, FI, FO) is the most degenerated situation, where it is obvious that no learning takes place. In Chapter 7, we use FFF environments as a baseline for this reason. On the other hand, the condition HHH (HT, HI, HO) is the less constrained, and a variety of different learning situations fit this profile. In HHH for instance, the density of $\mathscr{P}^*$ within $\mathscr{P}$ is unspecified, so it is unclear how difficult it is for learning agents to come across optimal policies. For instance, an agent learning within a very dense FHF IPL landscape possibly behaves like it would in a FFF profile. In other terms, the *scarcity* of P^* policies within $\mathscr{P}$ is an important additional qualifier of IPL profiles.

In practice, ML agents do not necessarily find an optimal policy P^*. Instead, they possibly fall back on the best policy found so far: $\hat{P}$. As a best-effort approximation of the elusive procedure $P^\star$, $\hat{P}$ is either almost as good as P^*, or its quality is much lower, depending on how the values $F\left(\hat{P}\right)$ and $F(P^*)$ compare together. For instance in the HLF condition, an agent that easily gets stuck on a good local optimum $\hat{P} \in \check{\mathscr{P}}$ possibly behaves like it would in a HHF profile. In other terms, the *quality* of non-optimal policies is another important property of the landscape not completely captured by the IPL profiles. Nevertheless, the above example shows that they constitute useful archetypes for concrete IPL situations to be compared to.

The core activity of IPL agents is to search policies. Consequently, the agent searching *method* is a major factor determining its trajectory within the landscape. For instance, agents supposed to solve the maximization problem with exact analytical methods are likely insensitive to the density of $\mathscr{P}^*$ within $\mathscr{P}$, and only the existence of optimal solutions, *e.g.*, $\check{\mathscr{P}}^* \neq \varnothing$, is meaningful to them. On the other hand, agents relying on plain random heuristics are likely sensitive to the density of $\mathscr{P}^*$ within $\mathscr{P}$. In this case, the more optimal policies, the more chances they have to eventually find one.

The same is not always true for local heuristics. With local heuristics, the exact *topology* of the landscapes plays a very important role. For instance, the hillier the landscape (the more non-optimal optima), the more difficult for the agents to find a P^*. On the other hand, agents easily converge if the landscape gradually slopes up towards global optima.

The landscape topology is important enough to skew the IPL profiles entirely. For instance in a HLF profile, if $\mathscr{P}$ is almost flat, with only one very narrow peak within $\ell\left(\bar{\mathscr{P}}\right)$,

the learning trajectory likely behaves as it would under FFF conditions. In this situation, we say that the profile is a rigorous HLF behaving like a FFF in practice. Alternately, if $\check{\mathscr{P}}$ shapes a very attractive slope towards a local sub-optimum $\hat{P} \in \check{\mathscr{P}}$, the trajectory likely behaves as it would under HHH conditions. In this situation, the profile is still a rigorous HLF, but we say it behaves like a HHH in practice. Once again, this demonstrates that the IPL profiles, although they are not sophisticated enough to predict the agents search trajectories, are useful as archetypes for practical IPL situations to be compared to.

In Chapter 6, we conduct an IPL experiment, in the less constrained HHH profile, using HLF and HHF environments as baselines. In Chapter 7, we experiment various profiles together, using FFF as a baseline, and refer to their HI, FO, *etc.* properties to discuss the observed results.

5.3.6 The Natural IPL Projections

The various factors described above (density of optimal policies, value of suboptima, landscape slopes) influence the learning trajectory of maximization agents in general, and not specifically IPL agents. The specificity of IPL is that the search space occasionally transforms with $(+i)$ and $(-i)$ events: $\bar{\mathscr{P}}$ becomes $\mathscr{P}$ and $\mathscr{P}$ becomes $\bar{\mathscr{P}}$ during the course of learning. This is were *transfers* need to be performed, in the sense of TL, and it has deep consequences for agents whose activity strongly relies on the search space topology, like agents performing local heuristics.

Whenever $(+i)$ happens, the policy $\bar{P} \in \bar{\mathscr{P}}$ currently tested is not meaningful anymore, and the agent search needs to be projected into some new location $P \in \mathscr{P}$. The nature of this projection matters, and where it should land depends on the landscape IPL profile. Assume for instance that the agent has converged towards $\bar{P}^* \in \bar{\mathscr{P}}^*$ before $(+i)$ happens. Then in every FT condition (see Figure 5.2), no policy in $\check{\mathscr{P}}$ yields better performance than $\bar{P}^*$, so the naive projection $P^* = \ell(\bar{P}^*)$ is a perfectly valid choice. In other terms, the agent should stick to its behaviour and ignore the new available input.

In HT conditions however, $\ell(\bar{P}^*)$ is a guaranteed suboptimal choice. Depending on the topology of $\mathscr{P}$, $\ell(\bar{P}^*)$ is also possibly a local optimum within $\mathscr{P}$, so there is a legitimate question whether to use ℓ as a projection to accommodate $(+i)$. On the one hand, ℓ guarantees not to land on the worst policy among $\mathscr{P}$, so the agent has a "jumpstart" advantage against a naive agent starting from anywhere within $\mathscr{P}$. On the other hand, ℓ possibly lands the agent into a maximization trap, where it remains stuck around the local suboptimal $\ell(\bar{P}^*)$ while the naive one eventually finds a global P^*. As a consequence, either the transfer uses the natural ℓ projection, and then care must be taken to trigger exploration again after the change event; or a more clever projection is used, but then the user needs *ad hoc* insights into the topology of $\mathscr{P}$ to construct it. This thesis defends that the natural projection is a good generic choice when there is no information about the topology. In the experiments presented in the next two chapters, the natural projection ℓ are used to perform the transfer in response to $(+i)$ events.

The situation is more complicated with $(-i)$ events since there exists no natural projection from $\mathscr{P}$ to $\bar{\mathscr{P}}$: information is lost. In other terms, learning agents cannot, in general, keep exhibiting the same behaviour when they are missing an input. This is of little importance in FI conditions (see Figure 5.2), because all diminished behaviours are equivalent. However, the question what projection to choose to accommodate $(-i)$ events is legitimate in every HI condition. For instance, in HHH landscapes, an ideal projection would directly land the diminished agent in the most promising region $\bar{\mathscr{P}}^* \subset \bar{\mathscr{P}}$, but this requires particular insights into the topology of $\mathscr{P}$ and $\bar{\mathscr{P}}$ that we cannot have at this level of generality.

Alternately, in FT conditions, there is a chance that the augmented agent behaviour is already part of $\ell\left(\bar{\mathscr{P}}^*\right)$. In this case, applying the reverse projection ℓ^{-1} is the best way to accommodate $(-i)$. In other terms, when the agent succeeds in the task at hand without making use of one input, then it should not be perturbed when this input is removed. To represent this, we define a particular class of projections called *almost-natural* projections from $\mathscr{P}$ back to $\bar{\mathscr{P}}$. A projection $\pi\colon \mathscr{P} \to \bar{\mathscr{P}}$ is almost-natural if:

$$\pi_{|\ell\left(\bar{\mathscr{P}}\right)} = \ell^{-1} \tag{5.61}$$

Almost-natural projections guarantee to keep the agent behaviour as-is if it was currently undergoing degenerated policies. In the experiment presented in the next two chapters, we use almost-natural projections π to perform the transfer in response to $(-i)$ events. We also defend that they constitute a good default, generic choice in the absence of *ad hoc* insights into the topology of $\bar{\mathscr{P}}$ and $\mathscr{P}$.

This section has investigated and drawn the principal characteristics that every high-level IPL problem should feature. In summary, projections between $\bar{\mathscr{P}}$ and $\mathscr{P}$ constitute the core IPL lever in accommodating $(+i)$ and $(-i)$ signature change events. When it comes down to particular learning situations, the more information users have about the topology of the search landscapes, the more accurate the corresponding IPL profile, and the more insightful the constructed projections.

5.3.7 Towards Generic IPL

While it is obvious that *ad hoc* algorithms perform better on the particular task they have been designed to address, ML has long been concerned with the construction of *generic* learning algorithms, that yield acceptable results in a variety of situations. Indeed, generic agents are the most useful when attempting to approximate an elusive procedure $P^\star$ which the user has no information about.

The previous sections have explained why *ad hoc* IPL projections are expected to yield good transfers because they exploit the landscape profile at hand. However, we have also noticed that there exists a class of natural projections, so-called because they preserve the current agent behaviour at best during a signature change event: the natural projection ℓ

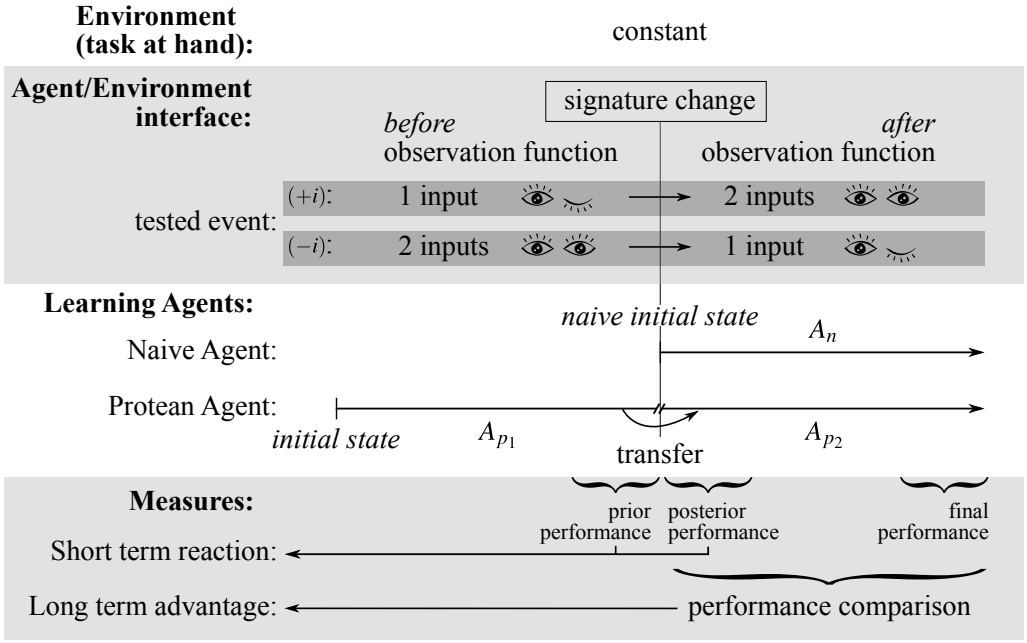


Figure 5.3: Global design of the experiments presented in chapters 6 and 7.

during $(+i)$, and the almost-natural projections $\not\pi$ during $(-i)$. This thesis defends that these natural projections are good candidates for accommodating IPL events in a generic way, when no or very few information is available about the IPL profile of the task.

The next two chapters address this claim with a couple of synthetic experiments. The first experiment (Chapter 6) verifies, in a simple OSL learning context, that a sophisticated RNN-based learning algorithm supports the natural projections without becoming unreliable, at least in HHH IPL landscapes, using HLF and HHF profiles as baselines. The resulting transfer makes the protean agent (A_{p_1}, A_{p_2}) more efficient than the naive A_n.

The second experiment (Chapter 7) addresses a more complicated learning context: RL, which features the credit assignment problem. We verify that natural projections can also be defined in other traditional learning algorithms, namely Q-Learning and Actor-Critic, and we qualify the advantage they provide depending on numerous properties of the task, and in particular, on its actual IPL profile.

While the two experiments differ much on the technical level and the learning situations they address, they are both designed according to the principles illustrated in Figure 5.3:

1. The agent is given a task to solve, which constitutes its environment. This task does not change during the agent lifetime.

2. The agent interface with the environment is defined in a flexible manner. Although their environment is the same, diminished agents perceive it with only 1 input, while augmented agents perceive it with 2 inputs. Augmented agents have complete information and can solve the task at hand.

3. Two transfer techniques are defined, so the agent can switch between an augmented or a diminished state without becoming undefined. The exact technique depends on the learning algorithm, but always constitutes a natural projection or an almost-natural projection.

4. A protean agent (A_{p_1}, A_{p_2}) is trained against the task, there are two different settings:

   - To test $(+i)$ event, the agent starts learning in a *diminished* state (A_{p_1}), then undergoes the change halfway through the learning process and is projected into an *augmented* agent (A_{p_2}) which resumes learning until the experiment is over.
   - To test $(-i)$ event, the agent starts learning in an *augmented* state (A_{p_1}), then undergoes the change halfway through the learning process and is projected into a *diminished* agent (A_{p_2}) which resumes learning until the experiment is over.

5. In parallel with A_{p_2}, a naive agent A_n is trained from scratch against the task. A_n has the same signature as A_{p_2}, but does not benefit from A_{p_1}'s transfer.

6. The learning traces of A_{p_1}, A_{p_2} and A_n are analysed with a small set of measures, to address:

   - Whether A_{p_2} has eventually succeeded in solving the task, by measuring *final performance*. While augmented agents are expected to always succeed after $(+i)$, it is not always possible for diminished agents to recover from the input loss after $(-i)$.
   - How A_n immediately reacts to the change event and the accommodating projection. To this end, the *prior performance*, right before the change is compared to *posterior performance*, right after the change. The immediate reaction to the change is supposed to be typically detrimental in the case of $(-i)$, but sometimes beneficial in the case of $(+i)$.
   - How A_{p_2} reacts to the change event on the long run, by comparing the whole A_{p_2} trace and A_n trace together. It is expected that A_{p_2} has an advantage over A_n because it benefits from the knowledge transferred from A_{p_1}.

The above protocol is declined and described in detail in chapters 6 and 7, along with the various addressed experimental conditions and extended analysis of the resulting learning traces. Chapter 8 concludes that natural projections are a good fit to basic generic IPL, while we also expound the limits of the approach.

Chapter 6

Experiment Input Protean Learning in Online Supervised Learning

This chapter tackles PL by addressing the basic viability of the natural IPL projections in a RNN-based agent, when $(+i)$ and $(-i)$ events happen during the course of learning in an OSL context. Are the generic projections compatible with the ANN technique? Do they succeed in performing the expected transfer?

To this end, we design and conduct a synthetic experiment where protean agents, modelled as (A_{p_1}, A_{p_2}), undergo signature changes and accommodate with natural projections, while naive agents A_n, as a baseline, resume the learning from scratch after the change events. The learning traces of all agents are studied with respect to the various conditions of the experiment, so we can dissect the numerous effects in play during IPL events accommodation. We conclude that the natural projections are not only useful to prevent protean agents from becoming undefined, but leave the RNN in a consistent state, and yield better results than the naive agents as they successfully avoid negative transfer.

These results, and the experimental design, were published in [Bonnici *et al.* 2020]. A preliminary version of the experiment, addressing less flexible and memory-less classes of functions, and the associated preliminary results were previously published in [Bonnici *et al.* 2019].

6.1 Design

The experiment is designed according to the principles described in Section 5.3.7.

Learning agents are trained in an environment that constitutes an OSL task. The task is chosen to mimic an idealized reinforcement situation like the imaginary roverbot described in Section 1.2, involving recurrence and streaming procedures, but without the credit assignment problem. The ideal policy $P^\star$ is known from the experimenter. This procedure is represented as a determination function that transforms input streams i into output streams $o^\star$ according to:

$$i \multimap (P^\star) \rightarrowtail o^\star \tag{6.1}$$

Note that $P^\star$ is recurrent, so it is non-Markovian and exhibits a memory of past values of i. An example realization $(i,\ o^\star)$ is illustrated in Figure 6.1.

Like in a SL context, the learning agents are supposed to approximate $P^\star$ given only a training set $\mathscr{T}$ with example realisations of $P^\star$:

$$\mathscr{T} = \{(i_n,\ o_n^\star)\}_{n\in\{1,\ ...,\ 1\,000\}} \tag{6.2}$$

With every $i_n \text{—}(P^\star)\twoheadrightarrow o_n^\star$. In the experiment, $\mathscr{T}$ is finite with only 1 000 elements. The streams processed are also finite and span over 128 timesteps.

Given clues successively drawn from $\mathscr{T}$, the agents are supposed to construct an approximation $\hat{P}$ of $P^\star$. During this process, they undergo either a $(+i)$ or a $(-i)$ event. The purpose of the experiment is to address that the natural IPL projections correctly accommodate the perturbation, and that the resulting protean agents $(A_{p_1},\ A_{p_2})$ perform better than a naive agent A_n.

An early version of this experiment was first described and the results were shown in [Bonnici *et al.* 2019]. In this early version, it only addressed $(+i)$ event with a weaker, less consistent parametrization. In this upgraded version, $(-i)$ is also addressed, and there are 4 controlled parameters supposed to qualify the IPL approach depending on the learning situation: ρ, κ, α and ξ. They address in particular the generic IPL profile HHH, and permit a comparison with two degenerated base profiles FHH and HLF. Here are the successive steps of the protocol described hereafter:

1. Generate synthetic inputs i_n with controlled correlation (κ) and noise (ρ).
2. Construct ideal behaviour $P^\star$ with controlled complexity (ξ).
3. Compute ideal outputs $o_n^\star$ with control of relevant input (α).
4. Construct the RNN-based learning agent A_{p_1} and make it explore $\mathscr{P}$ to find approximation of $P^\star$.
5. Perturb the agent with a signature change ($(+i)$ or $(-i)$), accommodate the change with a natural IPL projection to obtain A_{p_2}, resume learning.
6. Measure the advantage of A_{p_2} against the naive agent A_n.

All steps are described in the following sections.

6.2 Inputs Generation

In the reinforcement situation mimicked by the experiment, inputs i carry information supposed to help the agent in its task. This information is more or less predictable. For instance with the sticky roverbot, the position of the target is expected not to differ much between time steps if the user moves smoothly, but it is hard to predict if the user motion is fast and erratic.

As this predictability influences learning, we expect that it also influences the agent reaction to an IPL change. We assess it by generating various synthetic input data with a

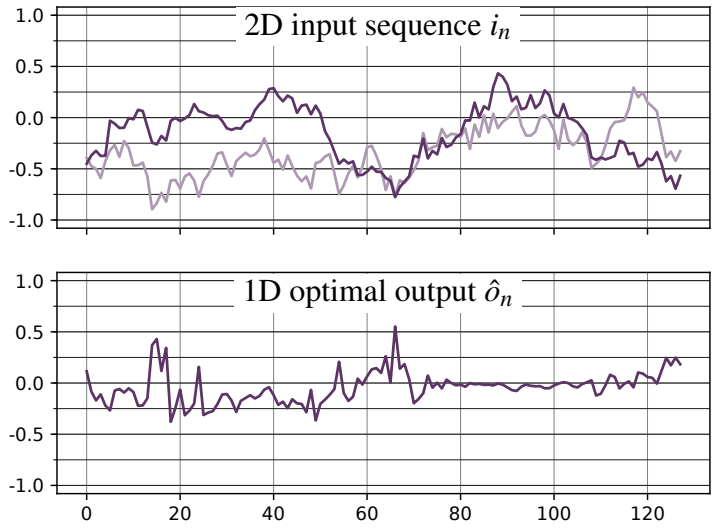


Figure 6.1: Example synthetic streams for the preliminary experiment. *Top:* Autocorrelated random input stream i_n with 2 correlated channels, generated with $\rho = 0.15$ (intermediate noise level) and $\kappa = 0.5$ (intermediate correlation level). *Bottom:* 1-channel output stream $\hat{o}_n$ computed from i_n with transformed random Legendre polynomial combination: $\alpha = 0.5$ (balanced usefulness of input channels), and $\xi = (2,3)$ (intermediate polynomial complexity). This situation is a case of the generic profile HHH.

controlled level of unpredictability, or "noise" ρ.

Besides, when several information channels are available, they also are more or less correlated together. For instance, when an infrared camera is plugged into the roverbot, the information it carries is essentially similar to the classical camera. However, when a battery sensor is plugged in, the new device produces original, decorrelated data instead. We expect this correlation level to influence the agent reaction to an input addition or deletion, and assess it by generating various synthetic data channels with a controlled level of correlation κ. The procedure is described below.

Each synthetic input i_n is generated as 2-channels (or "2D") data stream $(i_n^1,\ i_n^2)$. First, three reflected Gaussian random walks $i_n^a,\ i_n^b,\ i_n^c$ [Khanıev *et al.* 2001] are independently generated in the range $[-1,\ 1]$, with initial value uniformly chosen and standard deviation ρ. Then (see Figure 6.1, top) they are combined as:

$$i_n = \begin{pmatrix} i_n^1 \\ i_n^2 \end{pmatrix} = \begin{pmatrix} \kappa\, i_n^b + (1-\kappa)\, i_n^a \\ \kappa\, i_n^b + (1-\kappa)\, i_n^c \end{pmatrix} \tag{6.3}$$

The lower ρ, the more predictable i_n. The higher κ, the more redundant the two channels. In the experiment, three values are tested for ρ to assess the effect of input predictability:

- $\rho = 0.05$: The agent is given smooth inputs.
- $\rho = 0.15$: The agent is given noisier inputs.
- $\rho = 10$: The inputs are almost white noise.

Jointly, three values are tested for κ to assess the effect of channels redundancy:

- $\kappa = 0$: The agent is fed with two independent input channels.
- $\kappa = 0.5$: The two input channels carry correlated information.
- $\kappa = 1$: The two input channels are exactly identical.

The effects of IPL are later qualified with respect to these different tested situations.

6.3 Optimal Outputs Generation

The ideal policy chosen, $P^\star$, is a two-steps process (see Figure 6.1, bottom):

$$i_n \text{—}(Q)\text{→} i'_n \text{—}(R)\text{→} o^\star_n \tag{6.4}$$

The first step Q is to merge the two channels of i_n into one, without memory:

$$i'_n = \alpha\, i^1_n + (1-\alpha)\, i^2_n \tag{6.5}$$

with a mixing parameter $\alpha \in [0,\ 1]$ explained later. The second step R transforms i'_n into $o^\star_n$ with a parametrization permitting fine control of the task complexity.

In general, not all tasks are equally complex. We consider two reasons for this: First, some tasks require that the agent remembers past inputs so as to take best decisions. The older the past inputs, the more complex the task. Second, some expected outputs are not an easy, say linear, combination of the inputs. The less linear the combination, the more complex the task.

We expect the task complexity to influence IPL accommodation of signature changes. To assess this, we generate tasks with various complexity by choosing that R is constituted of a random multivariate polynomial M. The measure of complexity $\xi : (m,\ d)$ is twofold:

- The first component, m, is the depth of R memory, measuring how many past values of i'_n are used to compute one step:

$$o^\star(t+1) = M\big(i'_n(t-m+1),\ \ldots,\ i'_n(t)\big) \tag{6.6}$$

  In other terms, m corresponds to the dimension of M. The higher m, the more complex the task.
- The second component, d, is the degree of M. The higher d, the less linear M, and the more complex the task.

Generating random multivariate polynomials with controlled degree d is not straightforward, and three methodological obstacles need to be overcome:

1: A polynomial of degree d with random parameters sometimes does not behave as a full-degree polynomial. For instance, it can be almost linear. In this situation, the task complexity becomes over-estimated. To avoid this, M is defined as a random mixture of successive Legendre polynomials $(L_k)_{k\in\{0,\ \ldots,\ d\}}$ [Abramowitz and Stegun 1972], as each

L_k makes full use of its degree k:

$$M_l = \sum_{k=0}^{d} (-1)^{\sigma_k} w_k L_k, \quad \sum_{k=0}^{d} w_k = 1 \tag{6.7}$$

$$M\colon \begin{cases} [-1,\ 1]^m & \to [-1,\ 1] \\ (x_1,\ \ldots,\ x_m) & \mapsto \prod_{l=1}^{m} M_l(x_l) \end{cases} \tag{6.8}$$

The weights values w_k are conjointly drawn from a Dirichlet distribution so they sum to 1, and each random sign σ_k is drawn from $\mathcal{U}n(\{-1,\ +1\})$.

2: M is almost degenerated when the dominant weight w_d is lower than the others, resulting in the task complexity being overestimated again. To avoid this, the Dirichlet distribution concentration is set to $(1,\ \ldots,\ 1,\ 2)$ so w_d is on average twice higher than the other weights $(w_1,\ \ldots,\ w_{d-1})$.

3: As m and d increase, values of $M(x)$ become biased towards 0, so M is easily approximated with the null function, and the task complexity is over-estimated again. To avoid this, an additional transformation is applied to the values of $M(x)$ so as to stretch them away from 0. The transformation aims to restore the biased distribution to $\mathcal{U}n([-1,\ 1])$.

To this end, we first evaluate the distribution of $M(x)$: for each tested value of m and d, 200 random polynomials M are drawn and evaluated in 2 000 random points x. The distribution of all outputs $M(x)$ is estimated with a Gaussian Kernel Density Estimation (KDE) with Scott bandwidth selection [Jones *et al.* 1996]. We restore the distribution by using a 256-points linear interpolation approximation of the corresponding cumulative density function *CDF* as the stretching transformation. The final formula used for R is:

$$o_n^\star(t) = 2 \times CDF \circ M\Big(i'_n(t),\ \ldots,\ i'_n(t-m+1)\Big) - 1 \tag{6.9}$$

Three values of complexity ξ are tested, which correspond to properties of M:

- $\xi = (1,\ 1)$: M is linear and Markovian.
- $\xi = (2,\ 3)$: M is cubic and remembers last iteration.
- $\xi = (4,\ 3)$: M is cubic and reaches 4 steps back in time.

6.4 The IPL profiles

Not all inputs are equally useful. For instance, the battery level does not help the sticky bot when it only has to follow its target. We expect this relative usefulness to influence IPL reactions to signature change events. To assess this, we tune the value of α when mixing the 2 inputs channels of i_n into i'_n in equation (6.5). The higher α, the more useful the first input channel and not the other. Three values are tested for α:

- $\alpha = 0$: The channel i_n^1 is useless to solve the task at hand.
- $\alpha = 0.5$: The channel i_n^1 is useful, but not sufficient to solve the task.
- $\alpha = 1$: The channel i_n^1 contains all the information needed to solve the task.

The two parameters α and κ determine the eventual IPL profile of the task at hand. For instance, whenever $\alpha = 1$, the second information channel brings no useful information, so no behaviour in the augmented search space $\mathscr{P}$ yields better results than behaviours of the diminished search space $\bar{\mathscr{P}}$. As a consequence, the corresponding profile is LO (see Section 5.3).

Similarly, whenever $\kappa = 1$, the two information channels are strictly identical, so there is no behaviour that augmented agents exhibit that could not also be exhibited by diminished agents. As a consequence, the corresponding profile is also LO.

LO profiles are necessarily FT. And since diminished agents always get some relevant information when $\alpha = 1$ or $\kappa = 1$, it is always better to take it into account than to ignore it. As such, there are better diminished procedures than others and the profile is HI. In summary, degenerated tested situations with $\alpha = 1$ or $\kappa = 1$ all have HLF profiles.

On the other hand, the IPL landscape is FI whenever there is both $\alpha = 0$ and $\kappa = 0$, because diminished agent get no useful information to solve the task at hand, so there is no "better" diminished procedure than others. The landscape tested in this situation is FHH.

These two degenerated landscapes, FHH and HLF, constitute useful baselines to compare the generic landscapes, HHH, to.

6.5 Agent Structure and Learning Procedure

The agent approximates optimal behaviour $P^\star$ with its actual behaviour P. As a fixed parameter in the experiment, we choose a Recurrent Neural Network (RNN) to implement P. The explored search space $\mathscr{P}$ is therefore isomorphic to $\mathbb{R}^{Df}$, with Df the number of weights in the network.

RNNs are known to adjust arbitrarily complex recursive functions and infer arbitrarily numerous hidden states provided they contain enough internal states [Elman 1990; Schmidhuber 2015]. Therefore, they model any internal representation of the agent so that it progresses towards $P^\star$. As suggested by our preliminary study of $(+i)$ event [Bonnici *et al.* 2019], having not enough internal states results in the agent failing the task when the environment memory reaches too far back in time. We therefore use 3 standard Gated Recurrent Unit (GRU) cells as different network layers [Cho *et al.* 2014], with 6 internal states each, the last one being used as the network output. P produces the actual agent outputs according to $i_n \mathrel{—}(P)\mathrel{\rightarrow} o_n$.

The procedure A processes training examples by batches of 100 and updates weights of P with a stochastic gradient descent and 'adam' update rule [Kingma and Ba 2015] (learning rate $= 0.01$). On each batch, the Mean Squared Error $MSE(o,\, o^\star)$ is calculated as a loss. Convergence is achieved using pytorch [Paszke *et al.* 2017] for 1 000 iterations.

6.6 Signature Changes and Natural IPL Projections

In accordance with the principles described in Section 5.3.7, three learning passes are achieved on each run (see Figure 6.2). In the case of input addition $(+i)$:

1. The "first-form" protean agent A_{p_1} (left trace, in blue) is constructed with a diminished, 1D input signature. Its parameters are randomly initialized from the uniform distribution $\mathcal{U}n([-.01,\ .01])$. A_{p_1} is trained against $\mathcal{T}$, but only feeds from channel i_n^1 of the input streams, being blind to i_n^2.

   Note that in general, $P^\star$ cannot be reached in this case, because the agent is missing the full information to correctly approximate it. Besides, the 1D signature of P even differs from the 2D signature of $P^\star$. Only a reduction of $P^\star$: the closest 1D determination function, can be approximated.

2. The "second-form" protean agent A_{p_2} is then constructed with an augmented, 2D input signature. A_{p_2} is constructed from A_{p_1} according to the natural IPL projection ℓ: every initial RNN parameter in A_{p_2} is copied from the latest corresponding parameters in A_{p_1}, except for the necessary additional parameters, those supposed to weight the new input layer, which are set to zero. A_{p_2} is then trained against the whole training set $\mathcal{T}$ (green, right light trace), not ignoring the input channel i_n^2 anymore, so $P^\star$ can eventually be reached.

   This simple form of TL is considered transfer of "low-level" knowledge in [Taylor and Stone 2009], with an obvious mapping from the source task to the target task. With this construction, and consistently with the natural projection described in eq. (5.43), it is guaranteed that the augmented policy followed by A_{p_2} right after the $(+i)$ event is equivalent to the latest diminished policy followed by A_{p_1} right before the event. In simpler terms, the fresh A_{p_2} is constructed so it initially ignores the new input and behaves like A_{p_1}.

   No insight into the landscape of $\mathcal{P}$, or the corresponding IPL profile, were needed to realize this change, so the projection is generic. It is the very purpose of this experiment to verify that this IPL approach yields an acceptable transfer, or at least that it does not trigger unrecoverable perturbations within the ANN.

3. The "naive" agent A_n is constructed with a 2D signature. Its parameters are randomly initialized from the uniform distribution $\mathcal{U}n([-.01,\ .01])$. Then A_n is directly trained against the whole training set (black trace).

In the case of input deletion $(-i)$, the protocol is reversed: A_{p_1} has a 2D input signature and is able to see the whole dataset (blue, left trace), while A_{p_2} and A_n have a 1D input signature and are both blind to the second channel (red, black, right traces).

To realize this transfer, an almost-natural projection $\wp$ is operated on the RNN, with respect to eq. (5.61). The value of every weight parameter in A_{p_1} is copied into the

```
λ ← 0.01 # learning rate
NN, IP ← GRU_initialize() # Neural Network + Input Parameters
loop:
    (i*_b)_{b∈{1, ..., 100}}, (o*_b)_{b∈{1, ..., 100}} ← draw_batch(𝒯) # pick training samples
    if (+i) occurred:                # Patch algorithm to accommodate change events.
        IP ← concatenate(IP, 0)      # (+i): Append null column to input parameters matrix.
    if (−i) occurred:                # (−i): Remove column from input parameters matrix.
        IP ← drop_column(IP, 2)      # These constitute (almost-)natural projections.
    (ô_b)_{b∈{1, ..., 100}} ← NN(IP × i*) # compute network prediction
    loss ← MSE(ô, o*) # compute prediction error
    grad ← backpropagation(NN, IP, i*_b, loss) # compute local derivative of the network
    NN, IP ← adam(NN, IP, grad, λ) # learning step: update network parameters
```

Listing 6.1: Pseudocode for input signature change accommodation in traditional adam gradient descent for the GRU RNN fitting the batched training set $\mathscr{T}$. NN: GRU function and parameters (weights) except input layer. IP: learnable input-hidden weights of the first layer of the GRU (reset, update and new gates) [Cho *et al.* 2014], a matrix of size $(n_i,\ 18)$ in the experiment, with n_i the number of input channels (1 or 2), whose first dimension n_i is updated when the change event occurs.

corresponding weight in A_{p_2}, except for the ones without an equivalent weights, those corresponding to the lost input layer, which are dismissed. This construction results in a loss of information, but ensures again that the policy currently followed by A_{p_1} remains unchanged by the $(-i)$ event provided it was a degenerated policy with only zeroes on the discarded input layer. The purpose of this experiment is to verify that this IPL approach yields an acceptable generic transfer in this case.

In summary, the couple $(A_{p_1},\ A_{p_2})$ is our experimental OSL model of a PL agent, while the couple $(A_{p_1},\ A_n)$ represents a naive, non-PL agent. They both experience two elementary signature changes: $(+i)$ and $(-i)$, and their performances are compared together. 1 000 replicates of the experiment are run for each combination of experimental settings, and always with a freshly generated $P^\star$.

6.7 Measuring the Advantage of PL

Consistently with the experimental design described in Section 5.3.7, three measures are taken to assess the advantage of PL compared to the naive approach. They are computed on the learning curves $l\colon t \mapsto MSE(o(t),\ o^\star(t))$ (see Figure 6.2):

1: A *short-term* measure of transfer considers the loss jump occurring right after the signature change. It is the difference between the mean last 100 loss values of A_{p_1} and the mean first 100 of A_{p_2} (note that time is counted negatively prior to the event so it happens at $t = 0$):

$$\mathrm{IT} = \frac{1}{100}\left(\sum_{t=-100}^{-1} \log_{10}\left(l_{A_{p_1}}(t)\right) - \sum_{t=0}^{99} \log_{10}\left(l_{A_{p_2}}(t)\right)\right) \tag{6.10}$$

This "Immediate Transfer" measure is 0 when the event has no effect on learning (H), positive when it immediately improves learning (C, D), and negative when learning is perturbed by the event (A, B, E, F, G).

2: A *long-term* measure of the advantage is the mean *gain*:

$$\mathrm{LT} = \frac{1}{1000}\sum_{t=0}^{999}\log_2\left(\frac{l_{A_n}(t)}{l_{A_{p_2}}(t)}\right) \tag{6.11}$$

A measure $\mathrm{LT} = 1$ means that second-form agent A_{p_2} is twice better than the direct agent A_n on average. $\mathrm{LT} = 0$ means that they perform similarly.

3: A *last performance* measure is the mean last 100 loss values:

$$\mathrm{LP} = \frac{1}{100}\sum_{t=900}^{999}\log_{10}\left(l_{A_{p_2}}(t)\right) \tag{6.12}$$

When LP is below -2 (so the corresponding MSE is below 10^{-2}), we consider that the agent has succeeded in solving the task, like in all examples in Figure 6.2 except B.

6.8 Results

The measures obtained on each run are summarized in Figures 6.3 and 6.4. The various effects discussed here are named A, B, C, D, E, F, G and H, and illustrated in Figure 6.2 with example runs drawn from key conditions in Figures 6.3 and 6.4.

A generalized linear model was fitted on the data to address relevance of the observed variations. Predictions are represented as dots in Figures 6.3 and 6.4. All interactions were considered between experimental settings ρ, κ, α and ξ considered as factors (degrees of freedom: 80 919, residual stde: 0.7822). The effects discussed hereafter only rely on high significance contrasts with $p\text{-}value \leqslant 0.001$. Convergence and analysis of the model were achieved with R-Cran software [R Core Team 2020]. Interaction contrasts and their significance were calculated with the package phia [De Rosario-Martinez 2015].

As a general trend, even though the transfer sometimes has a negative short-term influence IT, the long-term score LT is positive on average. This reflects the advantage of the second-form agent A_{p_2} compared to the naive agent A_n. In other words, performing the transfer with the generic IPL projections is more beneficial than restarting the learning from scratch when the signature change occurs. This advantage needs to be qualified depending on the situation:

A. *IPL benefits from input redundancy*: In the $(+i)$ situation, it is expected that, when the new input carries information similar to the existing one, transfer makes A_{p_2} benefit from prior learning compared to A_n. This is confirmed by the data when $\alpha < 1$: the higher κ, the higher LT.

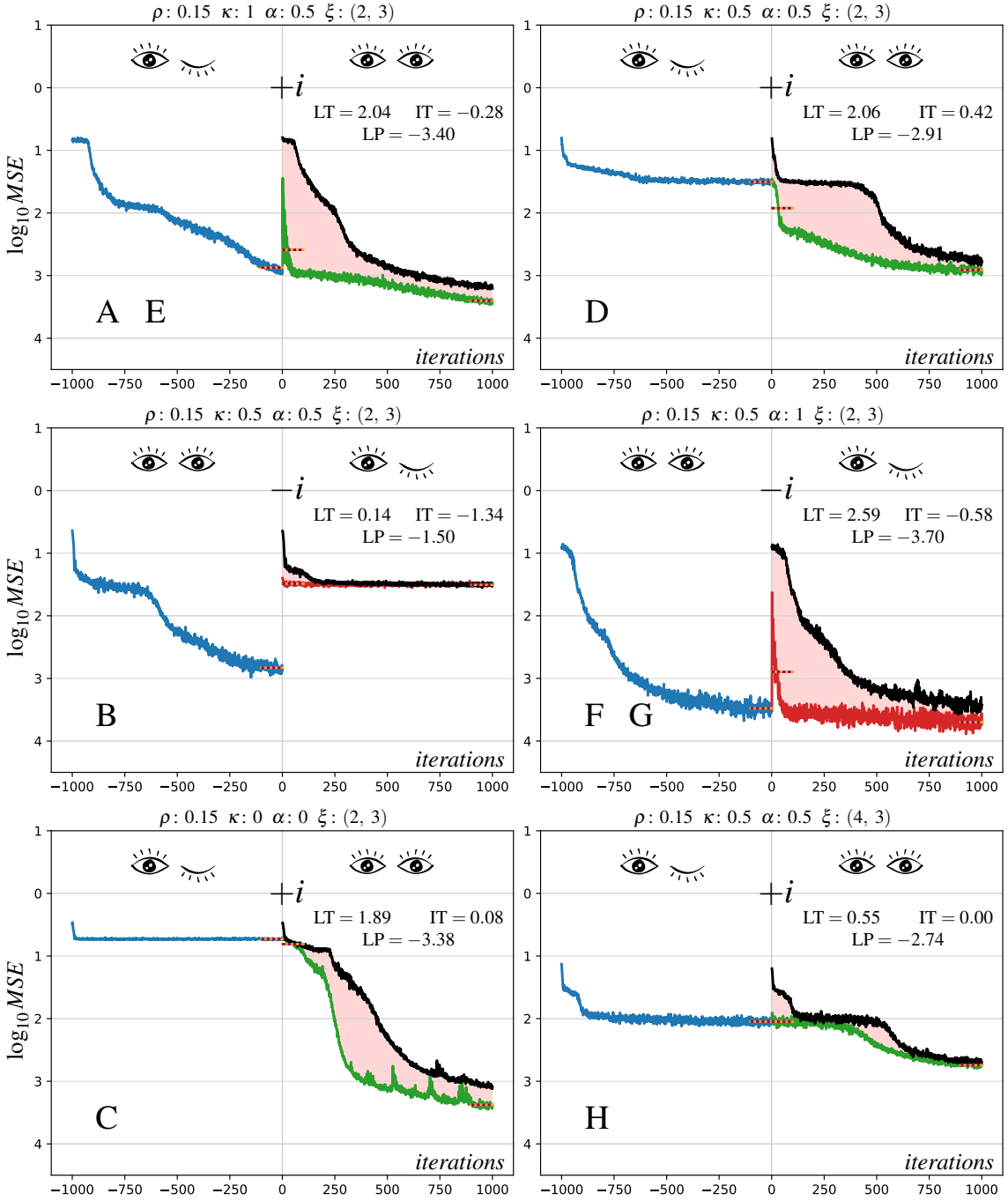


Figure 6.2: Learning curves l: evolution of MSE for key example individual runs in the experiment illustrating the observed effects A, B, C, D, E, F, G and H (Section 6.8). Filled areas illustrate the LT measure (7.85). Dotted bars illustrate the IT measure (7.87) and the LP measure (7.88). The eyes represent 1D/2D input signatures. The signature change event occurs at time 0.

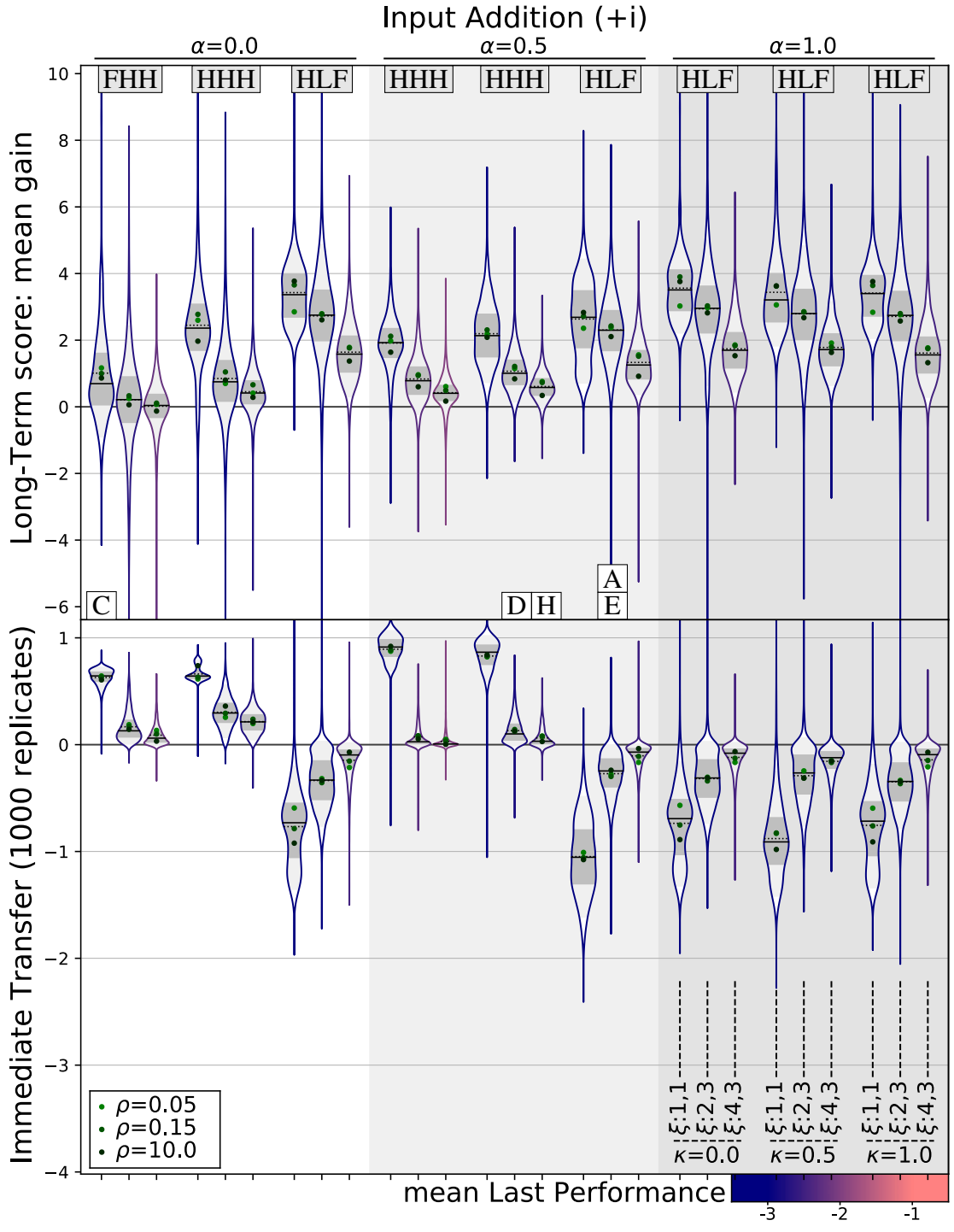


Figure 6.3: Violin plot: Comparison of metrics in the various experimental settings. Top pane: Long-Term advantage LT as defined in (7.85). Bottom pane: Immediate Transfer IT as defined in (7.87). Violin border shade indicates the mean value of the Last Performance LP as defined in (7.88). Violins are aggregated over ρ to ease readability, but statistical predictions of the metrics depending on ρ are represented as dots within the violins. *[Continued in Figure 6.4]*

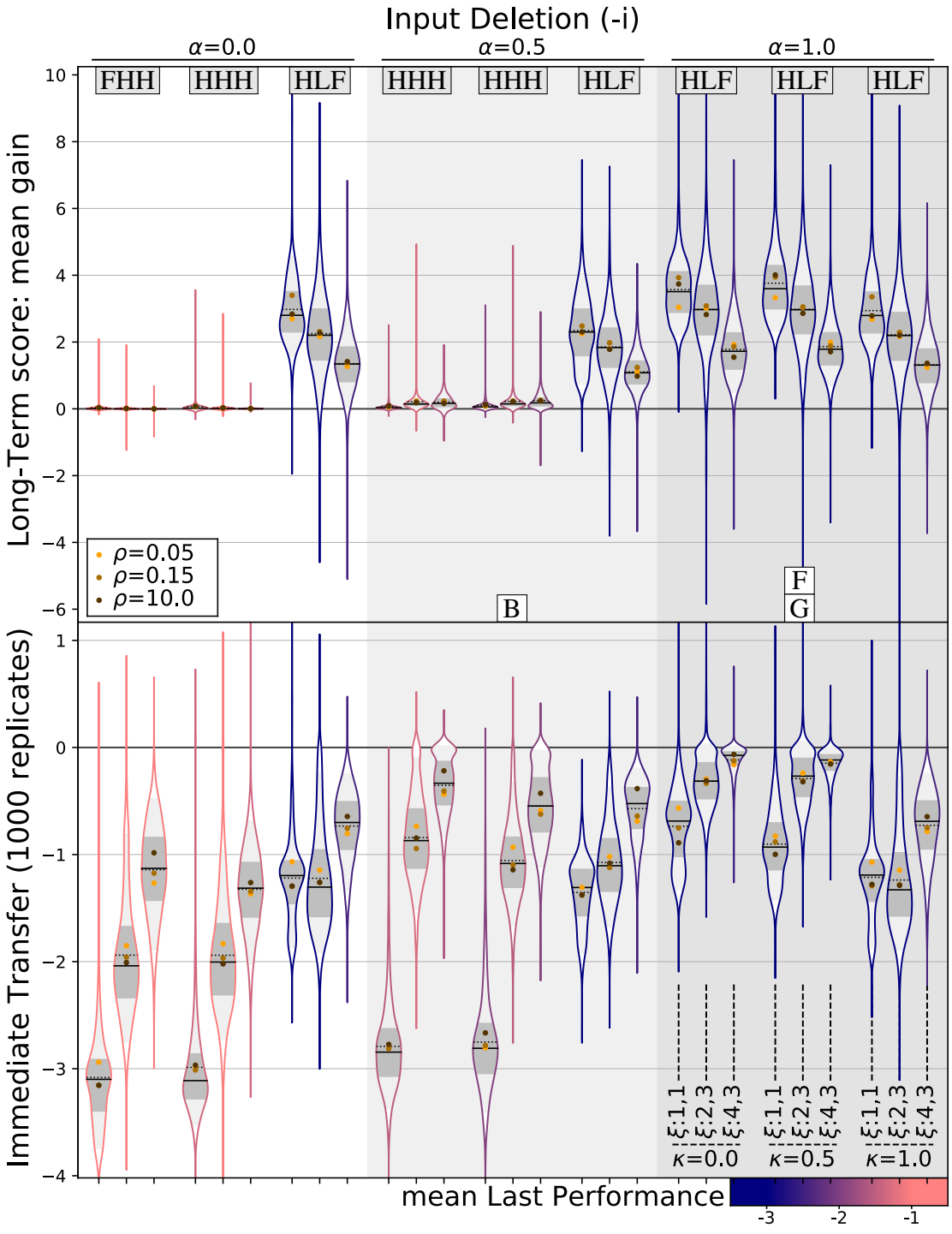


Figure 6.4: Same as Fig. 6.3 for the *input deletion* case. Solid lines represent median values, dashed line represent mean values. Grey areas in the violins represent 50% and 90% percentiles. White squared labels locate the conditions from which the examples in Figure 6.2 have been drawn. Grey square labels indicate the IPL profile corresponding to the tested conditions.

B. *Input loss yields failure*: In $(-i)$, it is expected that both agents fails when important input is removed. This is confirmed by the data when $\alpha < 1$, $\kappa < 1$: the LP scores are always above -2 on average. In this situation, performing the transfer or not does not make a strong difference, although LT scores are still significantly positive on average. B supports our abstract model of a task: the task is failed as expected when the agent has not enough information.

C. *IPL benefits from data structure*: In $(+i)$, it is expected that transfer is useless if the initial input carries no relevant information. But surprisingly, LT scores *are* positive even when $\alpha = 0$ and $\kappa = 0$ in FHH profiles, and higher if ρ is low. Our interpretation is that A_{p_1} still learns a correct representation/preprocessing of the data in this case, and that this knowledge benefits to A_{p_2}. This is further discussed in the next.

D. *IPL benefits from immediate transfer*: In $(+i)$, transfer is expected to be immediately beneficial. This is confirmed by the data when $\alpha < 1$ and $\kappa < 1$: IT scores are positive on average.

E. *There are negative transfer perturbations*: Effect D does not hold in $(-i)$ on the degenerated HLF tasks $\alpha = 1$ or $\kappa = 1$: IT scores are negative on average. In other words, when the new input is not useful to improve, the agent looses performance for a few iterations before figuring it out. Note that the long-term LT scores are still positive on average.

F. *IPL recovers with redundancy*: In $(-i)$, the removal perturbation is expected to be less severe when the inputs are somewhat redundant. This is confirmed by the data when $\alpha < 1$: the higher κ, the higher IT (note that IT is still negative on average). A_{p_2} transfers knowledge from the missing input to the remaining, similar input. LT score is even positive when $\kappa = 1$, and the task is still solved.

G. *IPL suffers from redundancy*: The effect F is reversed when $\alpha = 1$: the higher κ, the lower IT. Our interpretation is that, when only one of the two initial inputs is relevant, but the other is a copy of it, it takes longer for A_{p_2} to figure out that the lost information is still available in the remaining input. Note that the long-term LT scores are still positive on average.

H. *Complexity levels it off*: It is expected and observed that, the more complex the task (ξ), the less intense all the effects listed above. Indeed, when both A_{p_2} and A_n struggle to lower the error, their relative advantage becomes less clear. Still, LT remains positive on average.

In summary, we observe that IPL is overall beneficial on the long-term after a signature change. Most of these results were expected, especially in the generic HHH situations. However, it is worth noting that a short negative transfer perturbation (IT< 0) is observed

in special cases: obviously when an input is removed $(-i)$, but also when a useless one is added $(+i)$ in HLF situations, should it be redundant with existing inputs ($\kappa = 1$) or a simple distractor ($\alpha = 1$).

As with TL in general, there were no guarantees, prior to the experiment, that this negative effect would not overwhelm the whole learning process and take over the long-term scores LT during the runs. Now that we measure that the long-term effect is always positive on average, we are confident that the cost of negative transfer is, at worst, still better than the cost of restarting the learning from scratch. We have thus demonstrated that transfer with natural IPL projections works as expected, at least in this idealized OSL situation, and that the protean approach of learning is still conceivable.

A few effects were unexpected. C shows that transfer is beneficial even if the prior agent is completely unable to solve the task at hand, in a FHH situation. We suppose that it learns information about the structure of the data it is later fed from. Considering that, in this case, A_{p_2} and A_n both start with similar performance, but A_{p_2} learns more efficiently than A_n, we suggest that this phenomenon be studied as an alternative parameter initialization procedure. A parsimonious hypothesis is that A_{p_1} only learns to *ignore* the useless initial input in these cases.

In addition, the parameter ρ was found to strengthen the effect C: the more structured the data, the more the first-form agent had to transfer, even though it was always unable to solve the task itself. Considering this parameter in the general case, however, there was no clear influence of ρ on the reaction to the signature change event that we could identify. Agents fed with noisy data sometimes proved to be significantly better at IPL than agents fed with smooth data, and they sometimes proved to be significantly less good, although we could not understand why or when. Possible reasons for this include that the concept of "noise intensity" ρ is too weak as a representation of "data structure"; or that ρ only influences learning outside signature change events, so IPL would be insensitive to it; or that the influence of ρ on IPL is too complex to be figured out with the results of this simple experiment only.

According to TL theory [Taylor and Stone 2009], there can be various reasons for the second-form agent A_{p_2} advantage (see Section 4.2.1): either the agent gets a *jumpstart* benefit, or *learns faster*, or *learns better*. The LT metric (7.85) is an aggregated estimation of these 3 possible advantages, and IT (7.87) only measures jumpstart. As such, there is no way to distinguish all effects from each other with these measures. However, considering that many runs exhibit negative IT yet positive LT, we can confidently assert that the jumpstart effect is not the only benefit of IPL in this setup. This question was open in [Bonnici *et al.* 2019], and the effect C constitutes another argument in this favor.

The experiment presented in this chapter is idealized and abstract. It assesses the *sine qua non* condition that IPL learners can be developed at least in an easy situation like OSL, with the help of the generic natural projections.

The next chapter extends these preliminary results to a full-fledged RL situation, featuring all challenges of incremental learning, where the data appears piece by piece and not in full batch samples from a static training set [Losing *et al.* 2018]. In other terms, while the agent learning timeline (horizontal axis in Figure 6.2) and the input data stream timeline (horizontal axis in Figure 6.1) are not the same in the addressed OSL context, they need to line up in RL. With full-fledged RL, the credit assignment problem also has to be met [Sutton and Barto 2018]. Facing all these challenges to extend and complete the above results is the subject of the next chapter.

In particular, when it comes to comparing IPL with other baselines, the traditional RL benchmarks like mountain car and cart pole balancing [Sutton and Barto 2018] or Atari Games [Silver *et al.* 2016; Mnih, Kavukcuoglu, Silver, Rusu, *et al.* 2015] and 3D creatures [Gu *et al.* 2016; Lillicrap *et al.* 2015] cannot be directly used as they do not feature signature changes yet. We first need to adapt them in order to compare IPL to naive or alternate approaches. The first section of the next chapter is dedicated to the constitution of such an adequate benchmark.

Chapter 7

Experiment Input Protean Learning in Reinforcement Learning

The results shown in Chapter 6 address IPL natural and almost-natural projections in an OSL situation, within continuous HHH, HLF and HHF landscape profiles, and with deterministic agents. In this chapter, we conduct a similar experiment, with respect to the protocol specified in Section 5.3.7. but we extend its scope in three directions. First the learning situation is not OSL but full-fledged RL, so the agents have to face the credit assignment problem. Second, more landscape profiles are addressed, like HFF or FHF, using FFF as a baseline. Third, not only deterministic agents are tested, but also agents that visit a space of stochastic policies. For the experiment to remain tractable, we focus on a particular class of tabular RL environments, dedicated to addressing IPL, that we expound in section 7.1.3. The projections tested are always natural (against $(+i)$ events) or almost-natural (against $(-i)$ events).

To address PL in a RL situation, the agent needs to be embedded into a RL environment E. Instead of addressing one particular E, a whole class of environments is tested to qualify the IPL agent reactions depending on the environment properties. Like in the previous chapter, we restrict the study to IPL so we only consider $(+i)$ and $(-i)$ signature change events. We also restrict the analysis to "tabular" environments as described in [Sutton and Barto 2018].

There are two motivations for this restriction. First, tabular environments are tractable, so it is possible to analyse their properties. For instance, in the experiment, the true optimal policies P^* are known for each tested environment, and described in Section 7.1.7. Second, although they are hardly representative of full-fledged, large-scale environments faced by real world RL agents, the tabular environments constructed here are non-trivial and they exhibit diverse qualitative properties, corresponding to various IPL landscape profiles. This makes them useful as theoretical benchmarks.

The benchmark described in Section 7.1 is dedicated to addressing IPL in RL. In addition to testing various environments, two traditional RL methods, namely Q-Learning and

Actor-Critic with eligibility traces, are also tested to address IPL with both deterministic and stochastic approaches to policy approximation in RL.

7.1 A Tabular Benchmark for Input Protean Learning

To address IPL in RL, we design a class of synthetic RL environments that is both *minimal*, in the sense that it parsimoniously exhibits all properties relevant to IPL, and *diverse*, in the sense that it covers various possible learning situations. This section first describes tabular RL environments in general. Then, we explain the particular constraints enforced in our specific benchmark. Finally, we describe the environments actually used. We take time to express the benchmark properties in a generic way to encourage future extensions.

7.1.1 The transition function

A tabular RL environment is constituted of a set of *states* $\mathcal{S}$, a set of possible agent *controls* or *actions* $\mathcal{U}$, and a set of possible *rewards* or *feedbacks* $\mathcal{F}$. On each time step, the agent stands on a state $s \in \mathcal{S}$, and chooses an action $u \in \mathcal{U}$. As a response, the environment yields a feedback $f \in \mathcal{F}$ and moves the agent to a new state $s' \in \mathcal{S}$. This $(s,\ u,\ f,\ s')$ sequence is called a *transition*. Each environment is characterized by the transition procedure T that determines the next $(f,\ s')$ values for each possible state/action pair $(s,\ u)$.

In tabular cases, $\mathcal{S}$ and $\mathcal{U}$ are finite sets. As such, the transition procedure T is correctly specified with a finite labelled, directed multigraph. Each node of the graph represents a state $s \in \mathcal{S}$, while each arrow represents a possible transition resulting from picking a particular action $u \in \mathcal{U}$ in this state. Arrows source in the predecessor state s, they are labelled with the chosen action and the associated reward, *i.e.* $(u,\ f)$, and they target the associated subsequent state $s' \in \mathcal{S}$. For instance, with $\mathcal{S} = \{s_1,\ s_2\}$, $\mathcal{U} = \{u_1\}$ and $\mathcal{F} = \{0,\ 1\}$:

$$T:\ s_1 \underset{(u_1,\ 0)}{\overset{(u_1,\ 0)}{\rightleftarrows}} s_2 \circlearrowright (u_1,\ 1) \tag{7.1}$$

The example transition graph defined above is *non-deterministic*, because several different transitions are possible given one state/action pair. For instance, given $(s_2,\ u_1)$, it is unspecified whether the agent receives a feedback of 0 or 1, and whether it moves to state s_1 or s_2, because the two arrows exist.

Stochastic RL Environments RL traditionally features one particular class of non-deterministic environments: *stochastic* environments, also represented by Markov Decisions Processes. In stochastic environments, whenever there exists several possible transitions given one state/action pair $(s,\ u)$, the transition procedure T chooses among them

according to a probability distribution, *i.e.* $(f,\ s') \sim T(s,\ u)$. In this situation, $T(s,\ u)$ is a probability measure over the possible next transitions. For discrete feedback sets $\mathcal{F}$, T is therefore specified as:

$$T : \begin{cases} \mathcal{S} \times \mathcal{U} \to [0,\ 1]^{\mathcal{F} \times \mathcal{S}} \\ (s,\ u) \mapsto \begin{cases} \mathcal{F} \times \mathcal{S} \to [0,\ 1] \\ (f,\ s') \mapsto \mathbb{P}((f,\ s') \mid (s,\ u)) \end{cases} \end{cases} \tag{7.2}$$

$$\forall (s,\ u) \in \mathcal{S} \times \mathcal{U},\ \sum_{f \in \mathcal{F}} \sum_{s' \in \mathcal{S}} T(s,\ u)\big(f,\ s'\big) = 1 \tag{7.3}$$

In other terms, each arrow in the representative graph of T is associated with a probability value, marked after a colon symbol ':'. For instance, (7.1) would become:

$$T :\ s_1 \xrightarrow[(u_1,\ 0)\text{: } 0.9]{(u_1,\ 0)\text{: } 0.3,\ (u_1,\ 1)\text{: } 0.1} s_2 \circlearrowright (u_1,\ 1)\text{: } 0.7 \tag{7.4}$$

In this situation, picking action u_1 in state s_2 yields feedback $f = 1$ with 70% chances, and the agent stays on s_2, or it yields $f = 0$ with 30% chances, and the agent goes to s_1 instead. The transition function remains non-deterministic, but it is now completely specified.

Deterministic RL Environments RL environments are also possibly *deterministic*. In this case, there is always exactly one possible transition for each state/action pair. Deterministic environments can be considered degenerated stochastic environments where all probabilities yielded by the transition function are either 0 or 1 (see Box 1.1). Viewed another way, deterministic environments are degenerated non-deterministic environments where every node $s \in \mathcal{S}$ has exactly one outgoing arrow labelled with u for every $u \in \mathcal{U}$ in their representative graph, so there is no ambiguity left. For instance, (7.1) would become deterministic if there were 2 possible actions to pick instead of only 1, *i.e.* $\mathcal{U} = \{u_1,\ u_2\}$:

$$T :\ s_1 \xrightarrow[(u_1,\ 0)]{(u_1,\ 0),\ (u_2,\ 1)} s_2 \circlearrowright (u_2,\ 1) \tag{7.5}$$

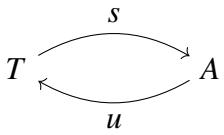
In this situation, the agent standing in s_2 has to pick action u_2 to get the reward and remain in s_2, because it would get no reward and move to s_1 if it picked action u_1 instead. Deterministic transition functions can be equivalently specified as a plain function:

$$T : \begin{cases} \mathcal{S} \times \mathcal{U} \to \mathcal{S} \times \mathcal{F} \\ (s,\ u) \mapsto (s',\ f) \end{cases} \tag{7.6}$$

Deterministic environments are simple and yield qualitatively different learning situations. As such, the benchmark developed in subsequent sections only contains deterministic environments. Under IPL context however, deterministic environments sometimes behave like non-deterministic environments from the agent perspective, so they are better addressed with stochastic agents. This is a major particularity of IPL discussed later. To understand this, we first need to specify how the agent perceives its environment.

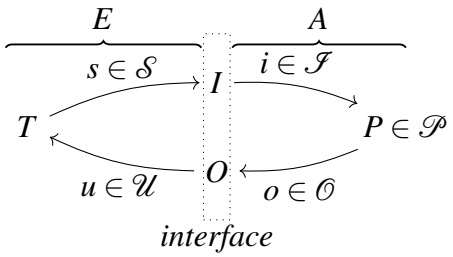
7.1.2 The Agent/Environment Interface

In the agent/environment retroaction loop, the agent procedure A is expected to produce actions u in response to incoming states s, like in this informal diagram:



But there is not always a 1-to-1 mapping between environmental states s produced by T and the inputs i actually received by the agent inner procedure P, or between the possible environmental actions u and the outputs o actually produced by P. In other terms, the agent does not perceive the environmental state *exactly*, and does not *exactly* produce all the actions expected by the environment. The signatures of P and T do not exactly match.

Signature Conversion The conversion of environmental states s into agent inputs i is represented by an *observation* procedure I, while the conversion of the agent outputs o into environmental actions u is represented by an *actuation* procedure O:



In the benchmark we describe hereafter, I is a deterministic function whose image is the set of every possible agent input $\mathscr{I}$:

$$I\colon \begin{cases} \mathcal{S} \to \mathcal{I} \\ s \mapsto i \end{cases} \tag{7.7}$$

The core property of IPL is that $\mathscr{I}$ is possibly modified during the course of learning. In the roverbot example, $(+i)$ occurs on a new sensor plug and $(-i)$ on a sensor removal, so

the set of possible environmental states $\mathcal{S}$ does not change, only I and $\mathcal{I}$. This is the case in our experimental benchmark, were both $I(t)$ and $\mathcal{I}(t)$ depend on time.

In the general PL context, the same principle holds for the set of possible outputs $\mathcal{O}$ and the actuation function O, but this falls outside the scope of IPL. In the environments we construct in the next, and for the sake of simplicity, the sets $\mathcal{O}$ and $\mathcal{U}$ are always the same, and O is the identity function:

$$\mathcal{O} = \mathcal{U} \tag{7.8}$$

$$O(o) = o = u \tag{7.9}$$

In other terms, in the experiment, the agent decides its actions exactly.

Given T, I, O, an initial state $s(0)$ and an inner agent policy P (described in Section 7.1.5), the agent/environment retroaction loop constitutes a fully specified dynamical system, yielding successive values of inputs, outputs, actions, feedbacks, and new states called together a *learning trajectory* J:

$$J = (i(t),\, o(t),\, u(t),\, f(t),\, s(t+1))_{t\in\mathbb{N}} \tag{7.10}$$

Input Signature Change Consistently with Section 5.3, $\mathcal{I}(t)$ results from the combination of various *input channels*, a tuple of domains $(\mathcal{I}_d(t))_{d\in\{1,\,\ldots,\,D(t)\}}$ whose Cartesian product constitutes $\mathcal{I}(t)$:

$$\mathcal{I}(t) = \prod_{d=1}^{D(t)} \mathcal{I}_d(t) \tag{7.11}$$

$$i(t) = (i_1(t),\, \ldots,\, i_D(t)) \tag{7.12}$$

With every $i_d(t) \in \mathcal{I}_d(t)$.

A $(+i)$ signature change event results in that a new channel $\mathcal{I}_{D(t)+1}$ is appended to the tuple, so the space of possible input values is extended to one more dimension. Conversely, a $(-i)$ signature change event results in that an existing channel $\mathcal{I}_d$ is removed from the tuple, so the space of possible input values is projected into a space missing the corresponding dimension d.

In the experiment (see Fig. 5.3), the learning trajectories last for $t_{\max} = 100\,000$ iterations. Each run only undergoes one of the above signature change events, either $(+i)$ or $(-i)$, at a specific time $t_{\text{change}} = 50\,000$. The agent signature is otherwise constant. Consistency with the notations used in Chapter 5, we use the symbols $\bar{I}$ and $\bar{\mathcal{I}}$ to refer to the *diminished* input signature (before $(+i)$ or after $(-i)$) and the symbols I and $\mathcal{I}$ to refer to the *augmented* input signature (before $(-i)$ or after $(+i)$).

7.1.3 A Minimal Environment

For the sake of parsimony, we attempt to construct our experimental benchmark from a minimal class of tabular environments where RL and IPL are still non-trivial.

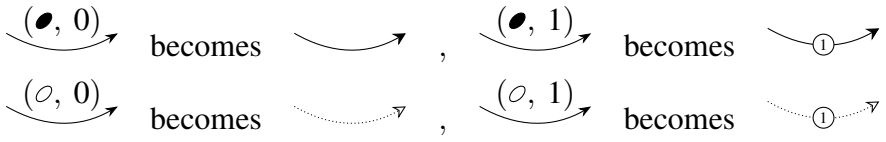
Minimal Signature The set of possible agent outputs $\mathcal{O}$ is chosen to only contain two values. As a consequence, on each step, the agent has a simple binary decision to make, say, "dark" or "light", which constitutes its very basic *dilemma*.

$$\mathcal{O} = \{\bullet, \circ\} \tag{7.13}$$

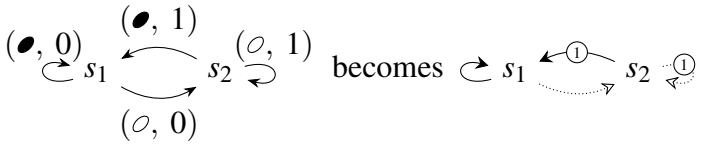
Similarly, the set of possible feedbacks is reduced to a binary set: either the agent gets null feedback, or it receives a unit reward. This is what makes the dilemma challenging.

$$\mathcal{F} = \{0, 1\} \tag{7.14}$$

The transitions graph T can therefore be simplified without ambiguity with the following graphical transformations. This is useful to specify the benchmark environments in subsequent sections:



In other terms, null feedbacks are elided and the arrow color indicates the chosen action $u = o$. For instance, the following two diagrams represent the same deterministic transition function:

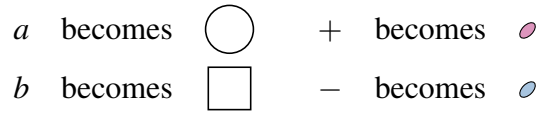


Minimal Signature Change Considering inputs, only two channels are used: $\mathcal{I}_1$ and $\mathcal{I}_2$, and each channel is also binary. To distinguish the values they carry, we call a and b the values carried by $\mathcal{I}_1$, while we use the symbols $+$ and $-$ to represent the values carried by $\mathcal{I}_2$:

$$\mathcal{I}_1 = \{a, b\} \tag{7.15}$$

$$\mathcal{I}_2 = \{+, -\} \tag{7.16}$$

The agent receives combinations of these values. To illustrate this, imagine that a and b convey shape information, while $+$ and $-$ convey color information:



The change events tested in the experiment are specified with respect to $\mathcal{I}_1$ and $\mathcal{I}_2$.

$$\bar{\mathcal{I}} = \mathcal{I}_1 \tag{7.17}$$

$$\mathcal{I} = \mathcal{I}_1 \times \mathcal{I}_2 \tag{7.18}$$

We refer to $\bar{\mathcal{I}}$ as the "1D" diminished signature, and to $\mathcal{I}$ as the "2D" augmented signature. 1D agents only distinguish 2 values: $\{a, b\}$, whereas 2D agents distinguish 4 different values: $\{(a, +), (a, -), (b, +), (b, -)\}$. For illustration, all agents in the experiment see the shapes, but 1D agents are insensitive to the color.

Minimal States The experimental environments are deterministic, so they contain no hidden state. The learning agent does not perceive the states s directly, but distinguishes 4 different values at most (in 2D). For all possible inputs values $i = I(s)$ to be useful, *i.e.* for I to be always surjective, environments have to contain exactly 4 different states s:

$$\mathcal{S} = \left\{ a^+ , a^- , b^+ , b^- \right\} \tag{7.19}$$

See for instance the 3 transitions functions T_A, T_B and T_C in Figure 7.1, which are part of the benchmark.

7.1.4 Diminished Projections of the Environment

The observation function I depends on the agent signature. In the 2D case, there is no loss of information, so the agent perfectly distinguishes the states from one another with its 2 channels:

$$I_{2D} : \begin{cases} a^+ \mapsto \left(\bigcirc, \ell\right) = (a, +) \\ a^- \mapsto \left(\bigcirc, \ell\right) = (a, -) \\ b^+ \mapsto \left(\square, \ell\right) = (b, +) \\ b^- \mapsto \left(\square, \ell\right) = (b, -) \end{cases} \tag{7.20}$$

In other terms, the augmented observation function is bijective:

$$I \colon \mathcal{S} \leftrightarrow \mathcal{I} \tag{7.21}$$

In the end, it accesses enough information to understand the full transition function.

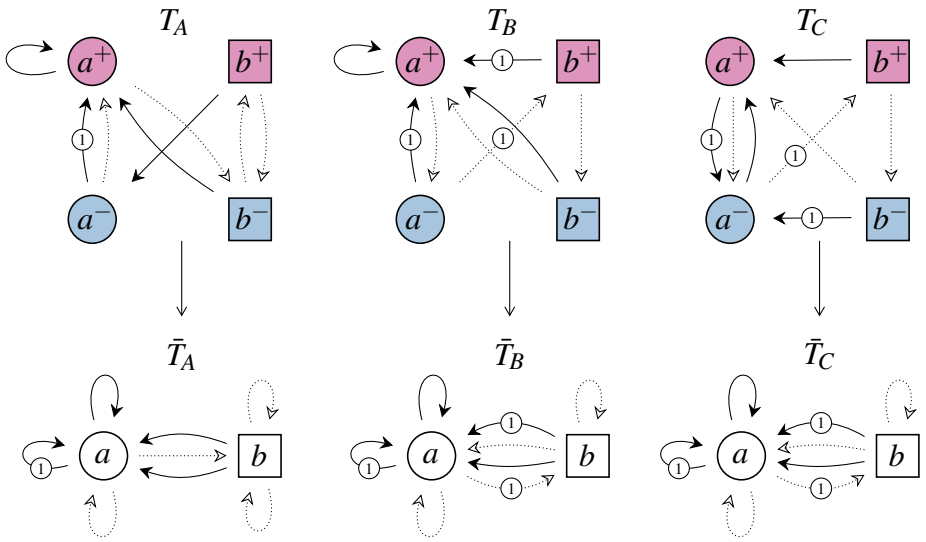


Figure 7.1: Example transition functions from the benchmark and their diminished projections.

However, in the 1D case, the agent only distinguishes the shape on the first channel:

$$I_{1D}: \begin{cases} a^+ \mapsto \bigcirc = a \\ a^- \mapsto \bigcirc = a \\ b^+ \mapsto \square = b \\ b^- \mapsto \square = b \end{cases} \tag{7.22}$$

In other terms, the diminished observation function is *not injective*:

$$\forall \bar{I}\colon \mathcal{S} \to \bar{\mathcal{S}},\ \neg\big(\bar{I}\colon \mathcal{S} \rightarrowtail \bar{\mathcal{S}}\big) \tag{7.23}$$

So 1D agents cannot distinguish a^+ from a^-, or b^+ from b^-: they are "missing a sensor". As a consequence, the best environment model they naively infer is a *non-deterministic* transition process $\bar{T}$, corresponding to a transformation of T where each pair of nodes with similar shape is collapsed into one node with undefined color (see Figure 7.1).

In $\bar{T}_A$, when receiving input a and choosing action $\bullet$, the 1D learning agent sometimes gets a reward, and sometimes not, depending on whether the true environment state was a^+ or a^-. Similarly in $\bar{T}_B$ or $\bar{T}_C$, when receiving input b and choosing action $\circ$, it sometimes navigates to a, and sometimes not, depending on whether the true environment state was b^+ or b^-. Note that T_B and T_C are projected into similar non-deterministic diagrams, that a naive deterministic agent cannot distinguish from each other.

Here, the source of non-determinism in $\bar{T}$ is not that the environment is stochastic, but that $\bar{T}$ contains *hidden states* (see Box 1.1). This is a key insight into IPL: signature change events like $(+i)$ or $(-i)$ trigger modifications of the observation function, resulting in that diminished agent face artefactual hidden states no matter the deter-

ministic nature of the environment. In other terms, the underlying Markov Decision Process (MDP), becomes a Partially Observable MDP (POMDP) [Spaan 2012]. As a consequence, the nature of the policy followed by the agent is decisive regarding its ability to learn the best behaviour, this is the object of the next section.

7.1.5 Policy Types

In the situation described above, the searched policy space $\mathscr{P}$ plays a central role. Of the three policy types described in Box 1.1 and Section 5.2.7 (deterministic, stochastic and recurrent), only recurrent policies like in eq. (5.36) can systematically represent the optimal behaviour in diminished $\bar{T}$ situations, and solve the underlying POMDP. Indeed, recurrent policies are equipped with a *memory*, so the agent can infer the environmental hidden state and reproduce it internally. For instance in $\bar{T}_A$, a recurrent agent possibly figures out that receiving input a and choosing action $\bullet$ yields a reward when previous input was b, but not when it was a, so it can adjust decisions based on this 1-step delayed information. In other terms, based on memory, the agent can correctly discriminate the transitions $(a^-,\ \bullet,\ 1,\ a^+)$ and $(a^+,\ \bullet,\ 0,\ a^+)$ from each other although they both appear as "$(a,\ \bullet,\ 0|1,\ a)$".

On the other hand, deterministic policies like in eq. (5.33) are not sophisticated enough to figure this out, as they only depend on current input. In $\bar{T}_A$, if a deterministic agent decides to pick $\bullet$ when receiving a because it leads to $(a^-,\ \bullet,\ 1,\ a^+)$, then it remains trapped forever in the $(a^+,\ \bullet,\ 0,\ a^+)$ transition loop and accumulates no reward on the long run.

A compromise is struck by non-recurrent stochastic policies like in eq. (5.34). Non-recurrent stochastic policies have no memory, but can reasonably enjoy the $(a^-,\ \bullet,\ 1,\ a^+)$ reward without remaining stuck forever in $(a^+,\ \bullet,\ 0,\ a^+)$, because there is always a chance that they escape with the other path $(a^+,\ \circ,\ 0,\ b^-)$ and get the reward again.

This experiment addresses how memory-less agents searching *deterministic* policies spaces $\mathscr{P}_{det}$ and non-recurrent *stochastic* policies spaces $\mathscr{P}_{sto}$ react to IPL signature change events $(+i)$ and $(-i)$. The case of *recurrent* policies $\mathscr{P}_{rec}$, featuring memory, is left for future works, and the benchmark environments constructed here will also be useful in addressing them.

In this experiment, diminished deterministic policies $\bar{P}_{det} \in \bar{\mathscr{P}}_{det}$ and augmented deterministic policies $P_{det} \in \mathscr{P}_{det}$ are specified as plain functions:

$$\bar{P}_{det}: \begin{cases} \bar{\mathscr{I}} \to \mathscr{O} \\ \bar{i} \mapsto o \end{cases} \qquad (7.24) \qquad\qquad P_{det}: \begin{cases} \mathscr{I} \to \mathscr{O} \\ i \mapsto o \end{cases} \qquad (7.25)$$

Here are two examples in the context of this benchmark:

$$\bar{P}_{det}: \begin{cases} a \mapsto \bullet \\ b \mapsto \circ \end{cases} \qquad (7.26) \qquad\qquad P_{det}: \begin{cases} (a,\ +) \mapsto \bullet \\ (a,\ -) \mapsto \circ \\ (b,\ +) \mapsto \circ \\ (b,\ -) \mapsto \bullet \end{cases} \qquad (7.27)$$

The example (7.26) represents a diminished agent that always chooses action ● when receiving input a, and action ○ when receiving input b. Example (7.27) shows an augmented agent that follows a more sophisticated policy because it distinguishes more states, but still being deterministic.

The natural injection of $\bar{\mathscr{P}}_{det}$ into $\mathscr{P}_{det}$, described in eq. (5.43), is straightforward: degenerated augmented policies simply ignore the second input channel. For instance, the above example $\bar{P}_{det}$ is projected into $\mathscr{P}_{det}$ as:

$$\bar{P}_{det}\colon \begin{cases} a \mapsto \bullet \\ b \mapsto \circ \end{cases} \implies \ell(\bar{P}_{det})\colon \begin{cases} (a,\ +) \mapsto \bullet \\ (a,\ -) \mapsto \bullet \\ (b,\ -) \mapsto \circ \\ (b,\ -) \mapsto \circ \end{cases} \tag{7.28}$$

In this situation, the augmented and diminished agents exhibit the same behaviour.

On the other hand, diminished stochastic policies $\bar{P}_{sto} \in \bar{\mathscr{P}}_{sto}$ and augmented stochastic policies $P_{sto} \in \mathscr{P}_{sto}$ are specified as parametrized probability measures:

$$\bar{P}_{sto}\colon \begin{cases} \bar{\mathscr{I}} \to [0,\ 1]^{\mathscr{O}} \\ \bar{\imath} \ \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(o = \bullet \mid i) \\ \circ \mapsto \mathbb{P}(o = \circ \mid i) \end{cases} \end{cases} \tag{7.29}$$

$$P_{sto}\colon \begin{cases} \mathscr{I} \to [0,\ 1]^{\mathscr{O}} \\ i \ \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(o = \bullet \mid i) \\ \circ \mapsto \mathbb{P}(o = \circ \mid i) \end{cases} \end{cases} \tag{7.30}$$

Here are two examples in the context of this benchmark:

$$\bar{P}_{sto}\colon \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = 0.4 \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 0.6 \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = 0.7 \\ \circ \mapsto \mathbb{P}(\circ \mid b) = 0.3 \end{cases} \end{cases} \tag{7.31}$$

$$P_{sto}\colon \begin{cases} (a,\ +) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\ +)) = 0.2 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\ +)) = 0.8 \end{cases} \\ (a,\ -) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\ -)) = 0.1 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\ -)) = 0.9 \end{cases} \\ (b,\ +) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\ +)) = 0.8 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\ +)) = 0.2 \end{cases} \\ (b,\ -) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\ -)) = 0.5 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\ -)) = 0.5 \end{cases} \end{cases} \tag{7.32}$$

The example (7.31) represents a diminished agent that chooses action ● with 40% chances when receiving input a, otherwise it chooses ○, and that chooses ● with 70% chances when receiving input b, otherwise it chooses ○. Example (7.32) shows again that augmented agents possibly follow more sophisticated policies because they distinguish more states, although they are still non-recurrent.

The natural injection of $\bar{\mathscr{P}}_{sto}$ into $\mathscr{P}_{sto}$, described in eq. (5.43), follows the same prin-

ciple. For instance, the above example $\bar{P}_{sto}$ injects into $\mathscr{P}_{sto}$ as:

$$\begin{aligned}
\bar{P}_{sto}\colon & \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = 0.4 \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 0.6 \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = 0.7 \\ \circ \mapsto \mathbb{P}(\circ \mid b) = 0.3 \end{cases} \end{cases} \\
\Longrightarrow\ \ell(\bar{P}_{sto})\colon & \begin{cases} (a,\,+) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\,+)) = 0.4 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\,+)) = 0.6 \end{cases} \\ (a,\,-) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\,-)) = 0.4 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\,-)) = 0.6 \end{cases} \\ (b,\,+) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\,+)) = 0.7 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\,+)) = 0.3 \end{cases} \\ (b,\,-) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\,-)) = 0.7 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\,-)) = 0.3 \end{cases} \end{cases}
\end{aligned} \tag{7.33}$$

In this situation, the augmented and diminished agents exhibit the same behaviour.

In addition to ℓ, there is also a natural injection m of deterministic policies into stochastic policies, since deterministic policies can be considered degenerated stochastic policies with only 0 or 1 probability measures:

$$m\colon \mathscr{P}_{det} \rightarrowtail \mathscr{P}_{sto} \tag{7.34}$$

For instance:

$$\begin{aligned}
P_{det}\colon & \begin{cases} (a,\,+) \mapsto \bullet \\ (a,\,-) \mapsto \circ \\ (b,\,+) \mapsto \circ \\ (b,\,-) \mapsto \bullet \end{cases} \\
\Longrightarrow\ m(P_{det})\colon & \begin{cases} (a,\,+) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\,+)) = 1 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\,+)) = 0 \end{cases} \\ (a,\,-) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (a,\,-)) = 0 \\ \circ \mapsto \mathbb{P}(\circ \mid (a,\,-)) = 1 \end{cases} \\ (b,\,+) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\,+)) = 0 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\,+)) = 1 \end{cases} \\ (b,\,-) \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid (b,\,-)) = 1 \\ \circ \mapsto \mathbb{P}(\circ \mid (b,\,-)) = 0 \end{cases} \end{cases}
\end{aligned} \tag{7.35}$$

This also holds for the diminished search spaces:

$$m\colon \bar{\mathscr{P}}_{det} \rightarrowtail \bar{\mathscr{P}}_{sto} \tag{7.36}$$

For instance:

$$\bar{P}_{det}\colon \begin{cases} a \mapsto \bullet \\ b \mapsto \circ \end{cases} \implies m(\bar{P}_{det})\colon \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = 1 \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 0 \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = 0 \\ \circ \mapsto \mathbb{P}(\circ \mid b) = 1 \end{cases} \end{cases} \tag{7.37}$$

In these situations, stochastic agents and deterministic agents exhibit the same behaviour.

Stochastic search spaces are therefore more general than deterministic search spaces, which is the reason they can expect better results when faced with diminished inputs like in examples shown in Figure 7.1. However, they still cannot infer the artefactual environmental hidden state because they are non-recurrent. The situation is summarized in the following diagrams:

$$\begin{array}{ccc} \mathscr{P}_{det} & \xrightarrow{m} & \mathscr{P}_{sto} \\ \ell \uparrow & & \uparrow \ell \\ \bar{\mathscr{P}}_{det} & \xrightarrow{m} & \bar{\mathscr{P}}_{sto} \end{array} \tag{7.38}$$

$$\mathscr{P}_{sto} \rightarrowtail \mathscr{P}_{rec} \tag{7.39}$$

Note that every injection relation is strict.

In the next, we combine the diagram (7.38) with diagram (5.58) to refine the IPL landscape profiles with respect to this relation between stochastic and deterministic search spaces. As a consequence, the environment IPL profile can be used as a parameter of the experiment no matter whether we address $\mathscr{P}_{sto}$ or $\mathscr{P}_{det}$. Before we do so, we need to enforce a few key properties of the benchmark.

7.1.6 Selecting Transition Functions

At this point, it is established that the environments used as a benchmark in the experiment all resemble the example environments T_A, T_B, and T_C shown in Figure 7.1. They are deterministic tabular environments with 4 states, expecting binary actions values $\mathscr{U} = \mathscr{O}$, yielding binary feedbacks $\mathscr{F}$, and producing either 1D (diminished) or 2D (augmented) observations on binary inputs channels combinations $\bar{\mathscr{I}} = \mathscr{I}_1$ or $\mathscr{I} = \mathscr{I}_1 \times \mathscr{I}_2$. This leaves room to numerous possible transitions functions (*e.g.*, T_A, T_B, T_C, ...), whose topology influences the learning and the agent reactions to IPL events. As a consequence, the benchmark we constitute does not consist in only a few transitions functions, but in every possible transition function fitting the above structure. As per the following count:

$$\begin{aligned} |\mathscr{T}| = \left|(\mathscr{S} \times \mathscr{F})^{\mathscr{S} \times \mathscr{U}}\right| &= (|\mathscr{S}| \times |\mathscr{F}|)^{|\mathscr{S}| \times |\mathscr{U}|} \\ &= (4 \times 2)^{4 \times 2} = 16\,777\,216 \end{aligned} \tag{7.40}$$

There are over 16 millions such transition functions.

However, not every $T \in \mathscr{T}$ is actually suited to RL. In particular, there are two properties of T we need to enforce in our benchmark:

- First, reinforcement is essentially a trial-and-error process. As a consequence, we cannot train RL agents within environments where it is impossible to reach back to an earlier state and try other policies. In other terms, we want T, as a Markov process, to be *recurrent*. Or we want T, as a directed graph, to be *strongly connected*. In the next, we filter out every environment in $\mathscr{T}$ not meeting this condition.

- Second, the RL task tackled in this experiment belongs to the *continuing* tasks category because it has no predefined duration or ending point [Sutton and Barto 2018]. As a consequence, we enforce that the reward eventually gathered by the agent within T only depends on its policy, and not on its starting state. In the next, we filter out every irrelevant environment from $\mathscr{T}$ regarding this condition.

In addition to the above restrictions, some transition functions within $\mathscr{T}$ are redundant because they are just symmetric copies of each other. In the next, we also filter outer out every redundant environment from the benchmark.

In the following few sections, we study $\mathscr{T}$ to structure the benchmark in three steps:

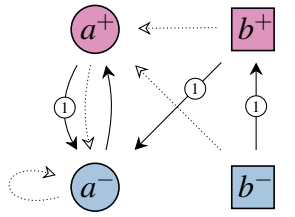
1. Filter out irrelevant transition functions from the benchmark according to the criteria above. This reduces its size to only 80 056 non-redundant relevant environments.

2. Use relations (7.38) and (5.58) to separate the benchmark into 8 distinct categories depending on their "joint" IPL profile. The environment joint profile becomes a parameter of the experiment.

3. Within each category, pick 100 transition functions at random to average the effect of IPL across various environments specificities.

7.1.6.1 Transitions Connectedness

Not every $T \in \mathscr{T}$ is relevant to RL. Consider the following transition function:

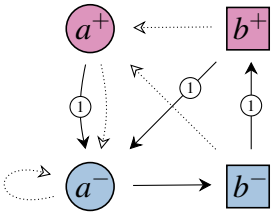
T : 

$\in \mathscr{T}$ (7.41)

Here T, as a Markov process, is not *recurrent*. This means that when the agent eventually reaches the subspace $\{a^+, a^-\}$, it will never be able to return to $\{b^+, b^-\}$ again, because there is no transition from a^+ or a^- to b^+ or b^-. This is a problem because RL is

essentially a *trial and error* process, where the learner needs to reproduce its experience until it finally figures out the best policy to solve the task at hand. In *episodic* tasks, recurrence is not a crucial property because the agent is guaranteed to always eventually start over [Sutton and Barto 2018]. But in the *continuing* tasks addressed here, if the environment does not allow the agent to reach back to an earlier state and have another try, then RL is not the right approach.

As a consequence, we restrict the benchmark to only contain the set of recurrent transition functions $\mathcal{T}_{\text{rec}} \subset \mathcal{T}$, *i.e.* the transition functions T such that for every $x,\ y \in \mathcal{S}$, there exists a chain of actions $(u_1,\ \ldots,\ u_n)$ that successively transitions from x to y. In other terms, the directed multigraph representing T must be *strongly connected*. For example, the following simple modification of T is recurrent and also part of the benchmark:

T :  $\in \mathcal{T}_{\text{rec}}$ (7.42)

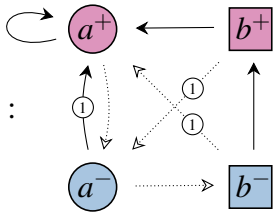
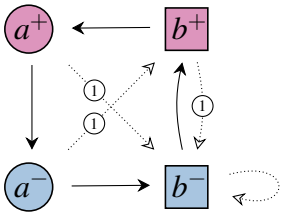
Indeed, b^- is reached from a^+ with the actions chain $(\circ,\ \bullet)$, a^+ is reached from a^- with the actions chain $(\bullet,\ \bullet,\ \circ)$, *etc.* In a continuing context with this environment, a RL learner is always able to reach back to a previous state and improve its policy again.

A careful screening of all transition functions in $\mathcal{T}$ yields the following count:

$$|\mathcal{T}_{\text{rec}}| = 5\,365\,248 \tag{7.43}$$

7.1.6.2 Transitions Symmetries

Not every transition function is unique from the learner agent perspective. For instance, consider the following two environments T_D and $\tilde{T}_D$:

T_D :  $\tilde{T}_D$:  (7.44)

The representation of T_D and $\tilde{T}_D$ look different, but they only differ by their labels, and the learning dynamics in either would always be the same in the experiment except for relabelling.

In other terms, among all automorphisms in $\mathcal{T}$, some are meaningless transformations $T \mapsto \tilde{T}$ that do not transform the environment from the agent perspective. These automorphisms are specified and studied in Annex A.1.

With these meaningless transformations correctly characterized, transition functions in $\mathscr{T}_{\text{rec}}$ are partitioned into C equivalence classes:

$$\mathscr{T}_{\text{rec}} = \bigcup_{c=0}^{C} \mathscr{T}_{\text{eq}}(c) \tag{7.45}$$

$$\forall c,\, c' \in \{1,\, \ldots,\, C\},\, c \neq c' \implies \mathscr{T}_{\text{eq}}(c) \cap \mathscr{T}_{\text{eq}}\big(c'\big) = \varnothing \tag{7.46}$$

Within an equivalence class $\mathscr{T}_{\text{eq}}(c)$, all environments are meaningless relabellings of each other, so they are redundant and only one of them needs to be tested in the benchmark. According to a deterministic procedure defined in Annex A.1, exactly one canonical transition function $T \in \mathscr{T}_{\text{eq}}(c)$ is picked within each class to finally constitute $\mathscr{T}_{\text{can}}$. Careful screening of $\mathscr{T}_{\text{rec}}$ yields the following count:

$$|\mathscr{T}_{\text{can}}| = C = 340\,136 \tag{7.47}$$

7.1.6.3 Sensitivity to Initial State

The possible agent policies $P \in \mathscr{P}$ differ by the actions they suggest in reaction to inputs, which implies that they trigger different transitions. As a result, they also differ by the feedback they collect. Here, we study the relation between P and the corresponding collected feedback $F(P)$ for the two classes of policies addressed in the experiment: deterministic policies $P_{det} \in \mathscr{P}_{det}$ and stochastic policies $P_{sto} \in \mathscr{P}_{sto}$.

First, consider augmented deterministic policies $P_{det} \in \mathscr{P}_{det}$, with these 3 examples:

$$P_{detA} : \begin{cases} (a,\, +) \mapsto \circ \\ (a,\, -) \mapsto \circ \\ (b,\, +) \mapsto \bullet \\ (b,\, -) \mapsto \circ \end{cases} \quad P_{detB} : \begin{cases} (a,\, +) \mapsto \bullet \\ (a,\, -) \mapsto \circ \\ (b,\, +) \mapsto \circ \\ (b,\, -) \mapsto \bullet \end{cases} \quad P_{detC} : \begin{cases} (a,\, +) \mapsto \bullet \\ (a,\, -) \mapsto \bullet \\ (b,\, +) \mapsto \circ \\ (b,\, -) \mapsto \circ \end{cases} \tag{7.48}$$

Environments in the benchmark are deterministic with finite numbers of states and transitions. So if the agent sticks to any P_{det}, its trajectory eventually cycles on itself forever.

For instance if the agent starts from initial state a^+ in T_D (A.1) and sticks to P_{detA}, the trajectory ends up trapped on the cycle $(a^-,\, \circ,\, b^-,\, \bullet,\, b^+,\, \circ)$ forever. This cycle is constituted of 3 transitions, and only 1 of them yields a reward. This enables a rigorous characterization of the absolute policy "value" $F_{T_D}(a^+,\, P_{detA}) = \frac{1}{3}$. The value represents the mean gain of the trajectory once trapped on a cycle. In its generic form:

$$F_T : \begin{cases} S \times \mathscr{P} \to\, [0,\, 1] \\ (s,\, P) \mapsto\, F_T(s,\, P) = \text{limit average reward} \end{cases} \tag{7.49}$$

Note that the initial state matters. For instance, there is $F_{T_D}(a^+,\, P_{detB}) = 0$, but also $F_{T_D}(b^+,\, P_{detB}) = \frac{1}{3}$, because the corresponding dynamical system is bistable with 2 different attractors $(a^+,\, \bullet)$ and $(b^+,\, \circ,\, a^-,\, \circ,\, b^-,\, \bullet)$, and they happen to yield different

mean reward 0 and $\frac{1}{3}$. Depending on the starting state, the agent ends up in one or the other, even though its policy, P_{detB}, is the same.

This property is undesirable in our context for three reasons. First, the RL task at hand is a *continuing* task, so the starting state is not expected to be encountered again on a regular basis. Second, the agents have no control over their starting state. Third, the state they happen to stand on whenever they start testing a new policy is contingent to their previous search trajectory. In shorter terms, we expect the agents to explore $\mathscr{P}$, and not $\mathscr{S} \times \mathscr{P}$. As a consequence, we restrict the benchmark so it only contains *starting state independent* transition functions $T \in \mathscr{T}_{\text{ssi}} \subset \mathscr{T}_{\text{can}}$ such that:

$$\forall s,\, s' \in \mathscr{S},\, \forall P \in \mathscr{P},\, F_T(s,\, P) = F_T\left(s',\, P\right) \tag{7.50}$$

Interestingly, the property (7.50) holds for every $P_{sto} \in \mathscr{P}_{sto}$ provided it holds for every $P_{det} \in \mathscr{P}_{det}$, because loosening a strict deterministic policy into a stochastic policy that allows the agent trajectory to occasionally escape deterministic cycles only results in connecting together cycles whose values are supposed to be the same. As a consequence, we only need to check the property for every $P_{det} \in \mathscr{P}_{det}$.

A careful screening of $\mathscr{T}_{\text{can}}$ yields that:

$$|\mathscr{T}_{\text{ssi}}| = 80\,056 \tag{7.51}$$

$\mathscr{T}_{\text{ssi}}$ constitutes the final benchmark used in our experiment.

For instance, the three example transition functions T_A, T_B and T_C specified in Figure 7.1 belong to $\mathscr{T}_{\text{ssi}}$. Note that (7.50) does not imply that the limit cycle is always the same regardless of the starting state, only that if there are several possible limit cycles, then they all have the same value. For instance in T_A, the reaction function P_{detC} (7.48) either leads to cycle $(a^+,\, \bullet)$ or cycle $(b^+,\, \circ,\, b^-,\, \circ)$, depending on the starting point, but both have value 0.

Note that the property (7.50) is weaker than the traditional requirement that RL Markovian environments be *ergodic* [Sutton and Barto 2018]. Ergodicity ensures that there exists a limit distribution of states and transitions in the corresponding process, but it constitutes a hypothesis on the environment structure that IPL profiles are indifferent to. T is ergodic if it is both recurrent and *aperiodic*. Section 7.1.6.1, has ensured that every transition function in the benchmark is recurrent, but $\mathscr{T}_{\text{ssi}}$ still contains *periodic* environments. In these environments, some states can only be attained on odd or event time steps, or multiples of 4, *etc.* However, their IPL profile is controlled.

Arguably, the restriction (7.50) reduces the benchmark diversity, since it rules out non-redundant, recurrent environments relevant for both RL and IPL. Consequently, we do not defend that this reduction should always be used when reproducing the benchmark, especially when testing learning agents able to control their starting state. This experiment uses the restriction for two reasons. First, every tested agent (see Section 7.2) is a simple reactive agent, either deterministic or stochastic. The addressed decision process (Q-Learning

or Actor-Critic) is sophisticated enough to "pick an action" regarding current input, but not to "navigate to a starting state of interest". Second, the restriction permits a separation of the benchmark into 8 *joint* landscape profiles described in the next section, which are used as experimental parameters.

7.1.7 Optimality Analysis of the Benchmark

In this section, we analyse the landscape of $\mathscr{P}_{det}$ and $\mathscr{P}_{sto}$ for every environment $T \in \mathscr{T}_{\text{ssi}}$ constituting the benchmark, with respect to the objective function F defined in eq. (7.49). This permits a categorisation of the landscapes into 8 joint IPL profiles.

7.1.7.1 Computing Mean Rewards

The environments in $\mathscr{T}_{\text{ssi}}$ are simple enough that the exact analytical profile of F can be computed within each of them. F is the limit distribution of the corresponding Markov Process, assuming uniformly randomized starting states $s(0)$, computed for every transition function in $\mathscr{T}_{\text{ssi}}$ with the help of the symbolic computation software Mathematica [Wolfram Research 2018]. Consider for instance a 1D agent trying the following diminished policy:

$$\bar{P}_{sto}\colon \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = p_a \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 1 - p_a \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = p_b \\ \circ \mapsto \mathbb{P}(\circ \mid b) = 1 - p_b \end{cases} \end{cases} \tag{7.52}$$

In the example environments T_A, T_B or T_C (Figure 7.1), we calculate that the mean expected future reward is:

$$F_{T_A}(p_a,\ p_b) = \frac{p_a\, p_b\, (p_a - 1)\, (p_b - 1)}{p_b^2\, (p_a - 2) + 2\, (1 + p_b - p_a)} \tag{7.53}$$

$$F_{T_B}(p_a,\ p_b) = \frac{p_a - p_b\, (p_a - 1)^2 - 1}{p_b + p_a\, (5 + p_a\, (p_b - 2) - 2\, p_b) - 4} \tag{7.54}$$

$$F_{T_C}(p_a,\ p_b) = \frac{p_b\, (p_b - 1)\, (p_a - 1)^2 - 1}{p_a\, (2 + p_b\, (p_b - 2)) - p_b\, (p_b - 2) - 4} \tag{7.55}$$

Functional analysis of this class of formulae yield the exact optimal policies sets $\bar{\mathscr{P}}^*$ and $\mathscr{P}^*$ for every environment in $\mathscr{T}_{\text{ssi}}$, along with the values of the maximum expected mean reward $\bar{F}^*$ and F^*.

For instance, in environment T_B, the maximum mean reward that a diminished 1D agent possibly attains with a stochastic policy from $\bar{\mathscr{P}}_{sto}$ is $\frac{2}{3}$. And the only way to attain it is to follow the degenerated policy where $p_a = 0$ and $p_b = 1$, so it turns out that the optimal behaviour is also possibly found by a deterministic agent exploring $\bar{\mathscr{P}}_{det}$. In T_C, the maximum possible value F^* is $\frac{1}{2}$, but any policy achieves it provided $p_a = 1$. This includes both policies from $\bar{\mathscr{P}}_{det}$ and policies exclusive to $\bar{\mathscr{P}}_{sto}$. In contrast, the maximum

expected reward in environment T_A is much lower, approximately 0.043[1], but is attained for $p_a \approx 0.660$ and $p_b \approx 0.416$, so no deterministic policy in $\bar{\mathscr{P}}_{det}$ achieves this score.

From these examples, it appears that the relation between optimal regions of $\mathscr{P}_{det}$ and optimal regions of $\mathscr{P}_{sto}$ is not straightforward. One consequence is that the IPL profiles of $\mathscr{P}_{det}$ and $\mathscr{P}_{sto}$ are not necessarily the same, even within the same environment. We therefore expect that the agents tested in the experiment undergo different learning experiences whether they explore deterministic or stochastic policy spaces. Eventually, both IPL profiles need to be specified as experimental context in the next.

7.1.7.2 Optimal Deterministic *vs.* Optimal Stochastic Policies

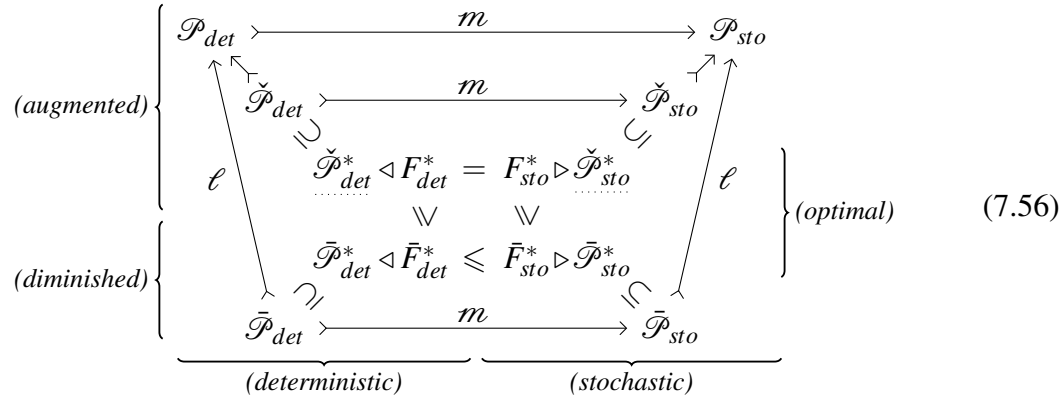
To expound the connections between $\mathscr{P}_{det}$ and $\mathscr{P}_{sto}$ profiles, we consider the set of relations shown in Section 5.3 regarding generic policy search spaces $\mathscr{P}$ again. These were summarized in diagram (5.58), reminded here:

$$\begin{array}{ll} \textit{(augmented policies)} & F^* \triangleright \check{\mathscr{P}}^* \subseteq \check{\mathscr{P}} \longrightarrow \mathscr{P} \\ & \wr\!\vee \qquad\qquad \nearrow_{\ell} \\ \textit{(diminished policies)} & \bar{F}^* \triangleright \bar{\mathscr{P}}^* \subseteq \bar{\mathscr{P}} \end{array}$$

The above does not yet distinguish deterministic policies from stochastic policies. Section 7.1.5, has also shown the following relations in diagram (7.38):

$$\begin{array}{ccc} \mathscr{P}_{det} & \overset{m}{\rightarrowtail} & \mathscr{P}_{sto} \\ {\scriptstyle \ell}\uparrow & & \uparrow{\scriptstyle \ell} \\ \bar{\mathscr{P}}_{det} & \overset{m}{\rightarrowtail} & \bar{\mathscr{P}}_{sto} \end{array}$$

Combining the two sets of relations in the context of our deterministic, tabular benchmark, we obtain the following connections between the various tested policies spaces:



(7.56)

The new relations appearing in diagram (7.56) are explained in the next.

[1] Exactly: the first root of a degree 5 polynomial, namely $x \mapsto 32x^5 - 192x^4 + 383x^3 - 216x^2 + 32x - 1$.

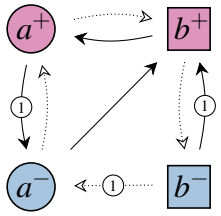
In environment T_A (Figure 7.1), every diminished deterministic policy yields the same reward $F_{T_A}(\bar{P}_{det}) = 0$ because all cycles have null value in T_A. However, with 1D stochastic policies, there are chances that the null cycles be sometimes escaped to undergo the $(a^-,\ \bullet,\ 1)$ transition without getting subsequently trapped forever in the $(a^+,\ \bullet,\ 0)$ cycle, yielding a positive mean expected reward. The formula (7.53) reveals that this mean reward is at most ≈ 0.043. In summary, it is possible that a diminished stochastic policy from $\bar{\mathscr{P}}_{sto}$ does better than the best diminished deterministic policy in $\bar{\mathscr{P}}_{det}$. Since every deterministic policy has an equivalent stochastic image $m(\bar{P}_{det}) \in \bar{\mathscr{P}}_{sto}$, this yields that in general:

$$\bar{F}^*_{det} \leqslant \bar{F}^*_{sto} \tag{7.57}$$

This is the reason why, as discussed in Section 7.1.4, there is an interest in tackling IPL situations with stochastic agents even within deterministic environments.

Such a situation is impossible with the augmented policy sets $\mathscr{P}_{sto}$ and $\mathscr{P}_{det}$, because the corresponding augmented observation function introduces no hidden state anymore. When the environment is finite and deterministic, any deterministic trajectory J induced by $P_{det} \in \mathscr{P}_{det}$ eventually ends up trapped on a cycle. The environment is therefore structured by the transition cycles that provide maximum mean reward. Consider for instance the following environment:

T_E :



(7.58)

The two cycles $(a^+,\ \bullet,\ a^-,\ \circ)$ and $(b^+,\ \circ,\ b^-,\ \bullet)$ yield the best possible mean reward: $\frac{1}{2}$. Among the set of every possible trajectories, the only best trajectories J^* are the ones that either stick to one of those optimal cycles, or occasionally switch from one optimal cycle to another. It is possible for deterministic policies in $\mathscr{P}_{det}$ to follow the first kind of optimal trajectories, here for instance with:

$$P_{det}: \begin{cases} (a,\ +) \mapsto \bullet \\ (a,\ -) \mapsto \circ \\ (b,\ +) \mapsto \circ \\ (b,\ -) \mapsto \bullet \end{cases} \tag{7.59}$$

Therefore, no trajectory in $\mathscr{P}_{sto}$ can yield better reward than the best deterministic policies in $\mathscr{P}_{det}$. In other terms:

$$F^*_{sto} = F^*_{det} \tag{7.60}$$

(7.57) and (7.60) complete the relations in diagram (7.56).

7.1.7.3 Joint IPL Profiles

A consequence of these non-trivial connections between stochastic and deterministic search spaces is that $\mathscr{P}_{det}$ and $\mathscr{P}_{sto}$ possibly belong to different IPL profiles, even within the same environment. For instance within environment T_A, the deterministic landscape matches the FI profile FHH because every $\bar{P}_{det}$ yields the same null reward, but the stochastic landscape matches the HI profile HHH because some $\bar{P}_{sto}$ are better than others. The environment T_A is therefore referred to with the joint profile: FHH-HHH.

Not every combination of IPL profiles into a joint profile is possible. There are a few particular constraints that we expound in Annex A.2. Before running the experiment, the joint profile of every environments in $\mathscr{T}_{\text{ssi}}$ is calculated. Careful screening of the benchmark yields the partition summarized in Table 7.1.

$\mathscr{P}_{det}$-$\mathscr{P}_{sto}$ joint profile	count
FFF-FFF	3 162
HFF-HHF	213
FHF-HHF	3 149
HHF-HHF	54 856
HLF-HLF	1 464
FHH-FHH	118
FHH-HHH	3 483
HHH-HHH	13 611
total $\lvert\mathscr{T}_{\text{ssi}}\rvert = 80\,056$	

Table 7.1: Joint profiles partitioning the $\mathscr{T}_{\text{ssi}}$ benchmark.

Some categories are missing from the benchmark, either because of the restrictions described in Table A.1, or because of the parsimonious nature of $\mathscr{T}_{\text{ssi}}$. Nevertheless, they are sufficiently diverse to address IPL in various learning situations. Degenerated profiles like FFF are useful as experimental baselines to address the more general cases like HHH.

100 transition functions are randomly drawn from each joint profile listed in table (7.1) to finally constitute the environments tested in the experiment.

7.2 The RL Agents

In the previous section, we have described the tabular RL environments E used in the experiment and their transition functions T. We have described the $(\mathscr{O}, \mathscr{I}(t))$ signature of the IPL learning agent within it, and also the policies search spaces $\mathscr{P}$ supposed to be explored by the agent for approximations $\hat{P}$ of the optimal policies P^*. The picture of the experiment run in this chapter is still incomplete, because it is unspecified how this exploration is performed. This section explains how the feedbacks values $f(t)$ produced by E are actually used by A to search $\mathscr{P}$.

7.2.1 Explored Search Spaces

The search procedure A is the heart of the *learning process*, and the object of RL science rather than strict IPL. This experiment tests 2 different, traditional learning procedures described in [Sutton and Barto 2018]:

- Q-Learning (QL), a simple traditional method well-suited for tabular environments. Agents undergoing QL explore $\mathscr{P}_{det}$ in search for good deterministic policies.
- Actor-Critic (AC), another traditional method searching $\mathscr{P}_{sto}$. To contrast with QL simplicity, we choose a sophisticated variant of AC featuring *eligibility traces*.

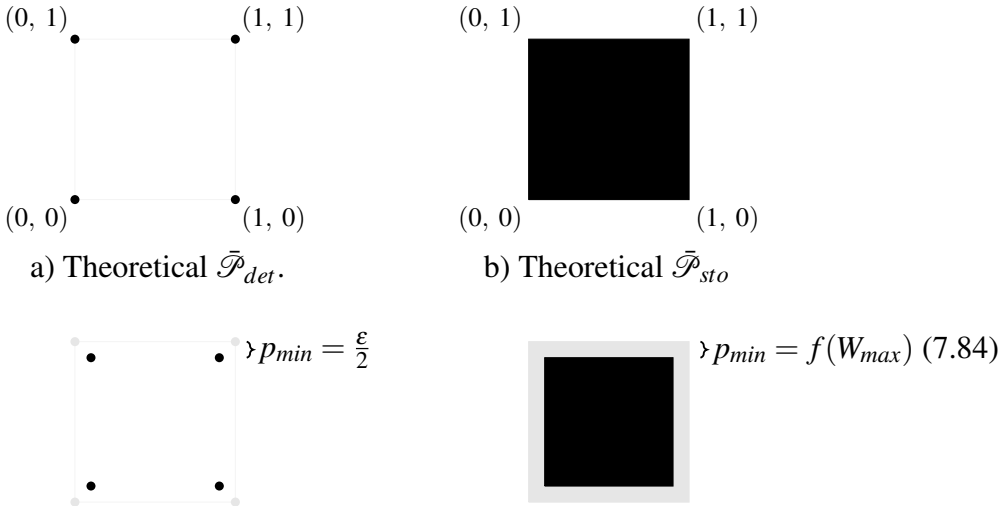
These procedures are well described in the literature, but it is yet unspecified how they are supposed to accommodate signature change events like $(+i)$ and $(-i)$. In the next, we explain the learning algorithm for QL and AC, but also the transfer projections we use to support IPL so the methods safely commute between $\bar{\mathscr{P}}$ and $\mathscr{P}$. In accordance with the principles expounded in Section 5.3, we only construct simple projections consistent with the natural projection $\ell : \bar{\mathscr{P}} \rightarrowtail \mathscr{P}$, and with the class of almost-natural reverse projections $\wp : \mathscr{P} \twoheadrightarrow \bar{\mathscr{P}}$. The core idea of the experiment is to verify that these projections provide satisfying results in a variety of IPL situations.

In the context of our benchmark, $\bar{\mathscr{P}}$ and $\mathscr{P}$ can be represented within $[0,\ 1]^{|\mathscr{I}|}$ hypercubes. For instance, there is a natural bijection between $\bar{\mathscr{P}}_{sto}$ and $[0,\ 1]^2$, represented in Figure 7.2.b. On the other hand, there is a natural bijection between $\bar{\mathscr{P}}_{det}$ and $\{0,\ 1\}^2$. $\bar{\mathscr{P}}_{det}$ corresponds to degenerated stochastic policies for which every associated probability measure is exactly 0 or 1. These lie on the corners of the hypercubes, as represented in Figure 7.2.a. Similarly, for augmented policies, there is a natural bijection between policies $\mathscr{P}_{sto}$ and $[0,\ 1]^4$, whose corners constitute $\mathscr{P}_{det}$ represented as $\{0,\ 1\}^4$.

Interestingly, the tested learning methods QL and AC do not exactly explore $\mathscr{P}_{det}$ and $\mathscr{P}_{sto}$, because they need to keep off the hypercubes boundaries to ensure continual exploration. In practice, QL explores a discrete set of stochastic policies as a proxy to $\mathscr{P}_{det}$, illustrated in Figure 7.2.c, and AC only visits a subset of non-degenerated stochastic policies as a proxy to $\mathscr{P}_{sto}$, illustrated in Figure 7.2.d. To ensure consistency of the results, we parameterize QL and AC so the corresponding offset p_{min} is the same in both approaches.

7.2.2 The Agent Objective

In section 7.1.7, we have categorized the environments based on their joint IPL profiles. To this end, every policy was evaluated by its future mean expected reward $F_T(P)$ (7.49). This value remains theoretical for three reasons. First, it assumes that the agent always sticks to the same policy, which is wrong in practice. Second, it is computed irrespective of the starting state, because of the property (7.50). Third, only a high-level analysis of the tabular environments makes its computation possible.



a) Theoretical $\bar{\mathscr{P}}_{det}$.

b) Theoretical $\bar{\mathscr{P}}_{sto}$

c) $\bar{\mathscr{P}}_{det}$ as explored by QL.

d) $\bar{\mathscr{P}}_{sto}$ as explored by AC.

Figure 7.2: Representation of diminished deterministic and stochastic search spaces in the benchmark: $\bar{\mathscr{P}}_{det}$ and $\bar{\mathscr{P}}_{sto}$. The first dimension of the squares represents the probability of picking action ● given input a, and the second dimension represents the probability of picking action ● given input b. Black regions in a) and b) represent the theoretical search spaces, while black regions in c) and d) represent the regions actually searched by the heuristic agents.

But the learning agent is blindly embedded into the environment, and does not know the underlying states or the structure of the transition function. Its current policy changes as it explores $\mathscr{P}$ and it is always standing on one particular state. As such, A cannot access or use F to guide its search within $\mathscr{P}$. Instead, it must approximate the value of each tested policy P with an *ad hoc* estimation of the future mean reward constituting its maximization goal. This estimation has to be computed from the feedbacks values $f(t)$ actually received during interaction with the environment. One common formal goal for the agent is the *discounted future gain* $G(P)$ [Sutton and Barto 2018]. The discounted gain is defined in stochastic environments under the assumption that the evaluated policy P is followed forever, and given a discount factor $\gamma \in [0,1[$:

$$G(P)(t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} f(t) \tag{7.61}$$

The parameter γ characterizes the learning procedure regarding the credit assignment tradeoff. When γ is 0, the agent only considers immediate rewards and is always greedy for the next transition. The higher γ, the more long-term the estimation and the agent picks actions that yield better future rewards even though they do not immediately pay off.

Interestingly, the value of $G(P)$ depends on time and involves the values of *future* feedbacks, which cannot be accessed by the agent either. The two RL approaches described in the next attempt to provide reasonable estimation of G, or the policies "values", based on past experience, so as to finally settle on acceptable policies $\hat{P}$.

7.2.3 Q-Learning

The first RL method tested in the experiment is QL. This is a search over the space of deterministic policies $\bar{\mathscr{P}}_{det}$ or $\mathscr{P}_{det}$, that aims at maximizing the expected discounted gain G (7.61) with an optimal policy $\bar{P}^*_{det}$ or P^*_{det}. To this end, the agent follows a stochastic policy that only differs from deterministic ones by a small probability measure $\varepsilon \in]0,\ 1[$ called *exploration rate*. For instance, instead of following $\bar{P}_{det}$ with:

$$\bar{P}_{det}\colon \begin{cases} a \mapsto \circ \\ b \mapsto \bullet \end{cases} \tag{7.62}$$

the QL agent undergoes the following stochastic policy:

$$\bar{P}_{sto}\colon \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = \dfrac{\varepsilon}{2} \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 1 - \dfrac{\varepsilon}{2} \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = 1 - \dfrac{\varepsilon}{2} \\ \circ \mapsto \mathbb{P}(\circ \mid b) = \dfrac{\varepsilon}{2} \end{cases} \end{cases} \tag{7.63}$$

as a proxy for $\bar{P}_{det}$. For this reason, QL is referred to as an *off-policy* method [Sutton and Barto 2018; Watkins 1989].

The parameter ε embodies the exploration *vs.* exploitation tradeoff. The higher ε, the better the exploration of the transition function. The lower ε, the lower the offset between the policies evaluated, $\bar{P}_{det}$or P_{det}, and the policies actually followed, $\bar{P}_{sto}$ or P_{sto} (see Figure 7.2.c).

While they follow the proxy policy, QL agents collect information from the environment to estimate one particular function: $Q : \mathscr{I} \times \mathscr{O} \to \mathbb{R}$, called a *value function*. Q is defined as the expected discounted gain (see eq. (7.61)) for a trajectory initialized by $(i,\ o)$ and then following an optimal policy P^*_{det} forever. To this end, the agent iteratively updates a table indexed by $I \times O$, initialized at 0, and holding every approximated value $\hat{Q}(i,\ o)$. The following update algorithm is run on each received $(i(t),\ o(t),\ f(t),\ i(t+1))$ quadruplet.

First, a critical value called the *temporal difference* is calculated:

$$\delta \leftarrow f(t) + \gamma \max_{o \in O} \hat{Q}\big(i(t+1),\ o\big) - \hat{Q}\big(i(t),\ o(t)\big) \tag{7.64}$$

δ is a measure of the prediction error, contrasting the current appreciation $\hat{Q}$ of the environment and the feedback value actually received $f(t)$.

In response to this error, the relevant estimated value is updated in the table, in the direction pointed by δ. A third method parameter called the *learning rate*, $\eta \in \mathbb{R}^{+*}$, spec-

ifies the magnitude of this update:

$$\hat{Q}\big(i(t),\, o(t)\big) \leftarrow \hat{Q}\big(i(t),\, o(t)\big) + \eta\, \delta \tag{7.65}$$

For instance, with $\eta = 0.1$, $\gamma = 0.9$ and a 1D agent exploring $\bar{\mathscr{P}}_{det}$, the following table transformation occurs after transition $(a,\, \circ,\, 1,\, b)$:

$$\begin{array}{c|cc} \hat{Q} & \bullet & \circ \\ \hline a & 0.1 & 0.2 \\ b & 1 & 0.8 \end{array} \quad \leftarrow \quad \begin{array}{c|cc} \hat{Q} & \bullet & \circ \\ \hline a & 0.1 & 0.37 \\ b & 1 & 0.8 \end{array} \tag{7.66}$$

The $\hat{Q}$ table determines the deterministic policy currently evaluated by the agent, according to the greediness rule:

$$\bar{P}_{det}\big(\hat{Q}\big) : \begin{cases} a \mapsto \underset{o \in \mathcal{O}}{\arg\max}\big(\hat{Q}(a,\, o)\big) \\ b \mapsto \underset{o \in \mathcal{O}}{\arg\max}\big(\hat{Q}(b,\, o)\big) \end{cases} \tag{7.67}$$

In the above example:

$$\bar{P}_{det}\big(\hat{Q}\big) : \begin{cases} a \mapsto \circ \\ b \mapsto \bullet \end{cases} \tag{7.68}$$

However, the algorithm must stick to the exploration principle so as not to get stuck on local optima. On each time step, there is a chance ε that a uniform random action be chosen instead of the one dictated by $\bar{P}_{det}$, thus the actually followed policy:

$$\bar{P}_{sto}\big(\hat{Q}\big) : \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid a) = \dfrac{\varepsilon}{2} \\ \circ \mapsto \mathbb{P}(\circ \mid a) = 1 - \dfrac{\varepsilon}{2} \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \mathbb{P}(\bullet \mid b) = 1 - \dfrac{\varepsilon}{2} \\ \circ \mapsto \mathbb{P}(\circ \mid b) = \dfrac{\varepsilon}{2} \end{cases} \end{cases} \tag{7.69}$$

For this reason, the set of policies actually visited during QL search is the one represented in Figure 7.2.c with $p_{min} = \frac{\varepsilon}{2}$.

The QL methods comes with theoretical guarantees. In the augmented, 2D situation, with the above update rule and the above policy combined, it is guaranteed that $\hat{Q}(t)$ eventually converges towards the true value function Q, and that $P_{det}\big(\hat{Q}(t)\big)$ eventually converges towards P^*_{det}. However, in diminished situations like the 1D case, this guarantee does not hold because the diminished observation function $\bar{I}\colon\, s(t) \mapsto i(t)$ is not injective anymore, so the environment admits hidden states. This makes it more difficult for IPL agent to accommodate diminished input situations.

The above algorithm becomes ill-defined when $(+i)$ and $(-i)$ events occur. For instance, the $\hat{Q}$ table belongs to $\mathbb{R}^{\mathcal{I}_1 \times \mathcal{O}}$ before the $(+i)$ event to represent diminished targeted policies in $\bar{\mathscr{P}}_{det}$, but it should belong to $\mathbb{R}^{(\mathcal{I}_1 \times \mathcal{I}_2) \times \mathcal{O}}$ after the $(+i)$ event, to represent augmented targeted policies in $\mathscr{P}_{det}$. These two sets are not isomorphic and $\hat{Q}$ cannot belong to both. As a consequence, the method needs to be adapted to accommodate IPL.

This is where the basic transfer projection that we wish to address in the experiment is introduced. When $(+i)$ occurs, we augment the $\hat{Q}$ table with a duplication of the relevant lines, according to:

$$\begin{gathered} \hat{Q} \leftarrow \hat{Q}', \quad \hat{Q} \in \mathbb{R}^{\mathcal{I}_1 \times \mathcal{O}}, \, \hat{Q}' \in \mathbb{R}^{(\mathcal{I}_1 \times \mathcal{I}_2) \times \mathcal{O}} \\ \forall (i_1, \, i_2), \, o \in (\mathcal{I}_1 \times \mathcal{I}_2) \times \mathcal{O}, \quad \hat{Q}'\big((i_1, \, i_2), \;\; o\big) = \hat{Q}(i_1, \, o) \end{gathered} \tag{7.70}$$

For instance:

$$\begin{array}{c|cc} \hat{Q} & \bullet & \circ \\ \hline a & 0.1 & 0.2 \\ b & 1 & 0.8 \end{array} \quad \leftarrow \quad \begin{array}{c|cc} \hat{Q}' & \bullet & \circ \\ \hline (a, \, +) & 0.1 & 0.2 \\ (a, \, -) & 0.1 & 0.2 \\ (b, \, +) & 1 & 0.8 \\ (b, \, -) & 1 & 0.8 \end{array} \tag{7.71}$$

This is consistent with the natural projection $\ell \colon \bar{\mathscr{P}}_{det} \rightarrowtail \mathscr{P}_{det}$, because the policy targeted by the agent after the transformation is the image by ℓ of the policy targeted before the transformation.

On a $(-i)$ event, the table is diminished instead with a mean aggregation of the relevant lines, according to:

$$\begin{gathered} \hat{Q} \leftarrow \hat{Q}', \quad \hat{Q} \in \mathbb{R}^{(\mathcal{I}_1 \times \mathcal{I}_2) \times \mathcal{O}}, \, \hat{Q}' \in \mathbb{R}^{\mathcal{I}_1 \times \mathcal{O}} \\ \forall i_1, \, o \in \mathcal{I}_1 \times \mathcal{O}, \quad \hat{Q}'\big(i_1, \, o\big) = \frac{1}{|\mathcal{I}_2|} \sum_{i_2 \in \mathcal{I}_2} \hat{Q}\big((i_1, \, i_2), \, o\big) \end{gathered} \tag{7.72}$$

This effectively results in information loss, for instance:

$$\begin{array}{c|cc} \hat{Q} & \bullet & \circ \\ \hline (a, \, +) & 0.9 & 0.1 \\ (a, \, -) & 0.7 & 0.8 \\ (b, \, +) & 0.2 & 0.6 \\ (b, \, -) & 0.5 & 0.4 \end{array} \quad \leftarrow \quad \begin{array}{c|cc} \hat{Q}' & \bullet & \circ \\ \hline a & 0.8 & 0.45 \\ b & 0.35 & 0.5 \end{array} \tag{7.73}$$

This is consistent with the almost-natural projections $\not{\rho} \colon \mathscr{P}_{det} \twoheadrightarrow \bar{\mathscr{P}}_{det}$, because the policy targeted by the agent before the change would not be transformed if it happened to be a degenerated image of ℓ. Note that this is not the only possible transformation consistent with almost-natural transformations. In future works, other $(-i)$ transformations will be tested like random aggregation or min/max aggregations.

```
s ← draw(𝒮) # random initial state          Q ← 0 # tabular action-state value function
η ← 0.1 # learning rate                     p_min ← 0.01 or 0.05 or 0.10 # minimal exploration rate
γ ← 0.9 # discount factor                   ε ← 2 × p_min # greediness parameter
Loop:
    With probability 1 − ε:
        a ← max_a Q(s, a) # pick greedy action
    else:
        a ← draw(𝒜) # pick random action
    Take action a, observe s′ and f′.
    If (+i) occurred:                       # Patch algorithm to accommodate change events.
        Q((s_1, s_2), ·) ← Q(s_1, ·)        # (+i): Duplicate every line in the value table.
    If (−i) occurred:                       # (−i): Aggregate pairs of lines.
        Q(s_1, ·) ← (1/|ℱ_2|) Σ_{s_2} Q((s_1, s_2), ·)   # These constitute (almost-)natural projections.
    δ ← f + γ max_a Q(s′, a) − Q(s, a) # calculate temporal difference
    Q ← Q + η δ # learning step: update value function approximation
    s ← s′ # step forward
```

Listing 7.1: Pseudocode for input signature change accommodation in traditional Q-learning algorithm with traditional $(s,\ a)$ notations for action/state. State is considered 2-dimensional $s = (s_1,\ s_2)$ and the $\mathbf{Q}$ table is either indexed by s_1 or $(s_1,\ s_2)$ depending on the underlying observation function arity. The highlighted structural transformations of the table correspond to the operations described in eqs. (7.70) and eq. (7.72).

These projections operations constitute the low-level *transfer* method that we address in our IPL experiment. The purpose is to verify that these simple accommodations of IPL work in practice for the traditional QL approach to RL, at least within the benchmark environments.

7.2.4 Actor-Critic

The second method tested in the experiment is AC. This is a search over the space of stochastic policies $\mathscr{P}_{sto}$ or $\bar{\mathscr{P}}_{sto}$. Instead of focusing on the value function approximation $\hat{Q}$, then deriving the optimal policy from it, AC attempts to directly approximate the optimal policy P^*_{sto} or $\bar{P}^*_{sto}$ with a gradient search over the policies landscape.

To this end, a *preferences* or "weights" table $W \in \mathbb{R}^{\mathscr{I}\times\mathscr{O}}$ is defined, in a fashion similar to the $\hat{Q}$ table, except that it does not contain approximations of the policy "values", but arbitrary numbers instead. The policy $P_{sto}(W)$ followed by the agent is the policy determined by the exponential softmax transformation of W:

$$P_{sto}(W)\colon i \mapsto \begin{cases} \bullet \mapsto \dfrac{e^{W(i,\ \bullet)}}{e^{W(i,\ \bullet)} + e^{W(i,\ \circ)}} \\ \circ \mapsto \dfrac{e^{W(i,\ \circ)}}{e^{W(i,\ \bullet)} + e^{W(i,\ \circ)}} \end{cases} \tag{7.74}$$

For instance in the diminished situation:

$$\begin{array}{c|cc} W & \bullet & \circ \\ \hline a & \text{-}0.8 & 0.2 \\ b & 0.6 & \text{-}1.3 \end{array} \Longrightarrow \bar{P}_{sto}(W): \begin{cases} a \mapsto \begin{cases} \bullet \mapsto \dfrac{e^{-0.8}}{e^{-0.8}+e^{0.2}} \approx 0.269 \\ \circ \mapsto \dfrac{e^{0.2}}{e^{-0.8}+e^{0.2}} \approx 0.731 \end{cases} \\ b \mapsto \begin{cases} \bullet \mapsto \dfrac{e^{0.6}}{e^{0.6}+e^{-1.3}} \approx 0.870 \\ \circ \mapsto \dfrac{e^{-1.3}}{e^{0.6}+e^{-1.3}} \approx 0.130 \end{cases} \end{cases} \tag{7.75}$$

In addition to W, AC uses another tabular function $\hat{V} \in \mathbb{R}^{\mathcal{I}}$ supposed to approximate the *state-value* function V. The definition of V resembles Q, but it does not involve the currently chosen action. The value $V(i)$ represents the expected future discounted gain for a trajectory starting from i then following the optimal policy forever. Finally, a running estimate of the next expected feedback, $\hat{f} \in \mathbb{R}$, is also used in this method. All values in tables W and $\hat{V}$ are initialized to 0, and so is $\hat{f}$.

Similarly to QL, entries in W, $\hat{V}$ and $\hat{f}$ are regularly updated whenever a new quadruplet $(i(t),\ o(t),\ f(t),\ i(t+1))$ becomes available. On each step of the learning procedure, the same typical value called *temporal difference* δ is first calculated:

$$\delta \leftarrow f(t) - \hat{f} + \hat{V}(i(t+1),\ o(t)) - \hat{V}(i(t),\ o(t)) \tag{7.76}$$

Then, the expected reward value is updated with learning rate η similar to the learning rate parameter in QL:

$$\hat{f} \leftarrow \hat{f} + \eta\,\delta \tag{7.77}$$

Regarding W and $\hat{V}$ updates, the learning rule is more complex because we choose a sophisticated variant of AC that uses *eligibility traces* to keep track of the most relevant values in the tables to update on each step. Eligibility traces are two associated tabular functions: $z_W \in \mathbb{R}^{\mathcal{I}\times\mathcal{O}}$ and $z_V \in \mathbb{R}^{\mathcal{I}}$, supposed to weight the entries in W and $\hat{V}$ most relevant to be updated next. Eligibility traces consider that a value is relevant to be updated not only if directly invoked by the current transition quadruplet, but also if it has *recently* been invoked by a former quadruplet. The further back in time the former quadruplet, the less relevant the update. This decay of relevance back in time is a geometrical decay defined with two additional method parameters λ_W and $\lambda_V \in [0,\ 1]$ called *eligibility decay rates*. They constitute another traditional accommodation of the credit assignment problem [Sutton and Barto 2018].

Eligibility traces are first set to zero. On each step, their past values decay and new values are bumped. This calculation involves the probability $\bar{p}$ that the agent had not taken

action $o(t)$ this time step:

$$\bar{p} \leftarrow \begin{cases} P_{sto}(i(t),\, \circ) \text{ if } o(t) = \bullet \\ P_{sto}(i(t),\, \bullet) \text{ if } o(t) = \circ \end{cases} \tag{7.78}$$

$$\forall (i,\, o) \in I \times O, \quad z_W(i,\, o) \leftarrow \lambda_W\, z_W(i,\, o) + \begin{cases} +\bar{p} \text{ if } o = o(t) \\ -\bar{p} \text{ if } o \neq o(t) \end{cases} \tag{7.79}$$

$$\forall i \in I, \quad z_V(i) \leftarrow \lambda_V\, z_V(i) + \begin{cases} 1 \text{ if } i = i(t) \\ 0 \text{ if } i \neq i(t) \end{cases} \tag{7.80}$$

The bump value in (7.80) corresponds to the gradient of the value $\hat{V}(i(t))$ with respect to the whole table $\hat{V}$. And the bump value in (7.79) corresponds to the gradient of the log-probability $\ln(P_{sto}(i(t),\, o(t)))$ with respect to the preferences table W. This classifies AC within the class of gradient search heuristics.

Once the eligibility traces have been updated, the values of $\hat{V}$ and W are updated according to:

$$\forall (i,\, o) \in I \times O, \quad \hat{V}(i,\, o) \leftarrow \hat{V}(i,\, o) + \eta_V\, \delta\, z_V(i,\, o) \tag{7.81}$$

$$\forall i \in I, \quad W(i) \leftarrow W(i) + \eta_W\, \delta\, z_W(i) \tag{7.82}$$

With dedicated learning rates η_W and $\eta_V \in [0,\, 1]$.

Like QL, the above procedure is guaranteed to converge towards P^*_{sto} in the augmented 2D case, due to the policy gradient theorem [Sutton and Barto 2018], but not in the diminished 1D case due to the presence of hidden states in the environment.

Like QL, AC agents enforce exploration of the environment and the policies space by ensuring that the preferred action is not systematically taken on each step, and that the other, non-preferred action is sometimes chosen instead. In QL, the non-preferred action is taken with probability $p_{min} = \frac{\varepsilon}{2}$ on each step. In AC, this probability depends on the preference table according to (7.74).

Consequently, when AC agents converge towards these particular, degenerated deterministic policies $P_{det} \in \mathscr{P}_{sto}$, their exploration rate p_{min} tends towards zero. This makes QL and AC not straightforward to compare in the experiment. To alleviate this phenomenon, the above AC procedure is modified to maintain a base level of exploration throughout learning. The last update operation (7.82) is completed with a clipping operation so the preferences magnitude never exceeds one limit value W_{max}:

$$\forall i \in I \times O, \quad W(i) \leftarrow \text{sign}(W(i)) \min(|W(i)|,\, W_{max}) \tag{7.83}$$

This way, the explored policy never tends towards strictly deterministic policy, as there is

$\mathbf{W} \leftarrow 0$ # preferences table parametrizing policy: $\pi(a|s,\ \mathbf{W}) = e^{\mathbf{W}(s,\ a)} / \sum e^{\mathbf{W}(s,\ \cdot)}$ (eq. (7.74))
$s \leftarrow \text{draw}(\mathcal{S})$ # random initial state $\quad \mathbf{V} \leftarrow 0$ # tabular state value function
$\mathbf{z}_V,\ \mathbf{z}_W \leftarrow 0$ # eligibility traces $\quad \hat{f} \leftarrow 0$ # mean reward estimation
$\lambda_V,\ \lambda_W \leftarrow 0.9$ # trace decay rates $\quad p_{min} \leftarrow 0.01$ or 0.05 or 0.10 # minimal exploration rate
$\eta,\ \eta_V,\ \eta_W \leftarrow 0.1$ # learning rates $\quad W_{max} \leftarrow \frac{1}{2}\ln\left(\frac{1}{p_{min}} - 1\right)$ # preferences bound (eq. (7.84))
Loop:
    $a \leftarrow \text{draw}(\pi(\cdot|s,\ \mathbf{W}))$ # pick action according to current policy
    Take action a, observe s' and f'.
    If $(+i)$ occurred: # Duplicate lines in tables (natural projection).
        $\mathbf{W}((s_1,\ s_2),\ \cdot) \leftarrow \mathbf{W}(s_1,\ \cdot) \quad \mathbf{V}((s_1,\ s_2)) \leftarrow \mathbf{V}(s_1)$
        $\mathbf{z}_W((s_1,\ s_2),\ \cdot) \leftarrow \mathbf{z}_W(s_1,\ \cdot) \quad \mathbf{z}_V((s_1,\ s_2)) \leftarrow \mathbf{z}_V(s_1)$
    If $(-i)$ occurred: # Aggregate pairs of lines in tables (almost-natural projection).
        $\mathbf{W}(s_1,\ \cdot) \leftarrow \frac{1}{|\mathcal{F}_2|}\sum_{s_2} \mathbf{W}((s_1,\ s_2),\ \cdot) \quad \mathbf{V}(s_1) \leftarrow \frac{1}{|\mathcal{F}_2|}\sum_{s_2} \mathbf{V}((s_1,\ s_2))$
        $\mathbf{z}_W(s_1,\ \cdot) \leftarrow \frac{1}{|\mathcal{F}_2|}\sum_{s_2} \mathbf{z}_W((s_1,\ s_2),\ \cdot) \quad \mathbf{z}_V(s_1) \leftarrow \frac{1}{|\mathcal{F}_2|}\sum_{s_2} \mathbf{z}_V((s_1,\ s_2))$
    $\delta \leftarrow f - \hat{f} + \mathbf{V}(s') - \mathbf{V}(s)$ # calculate temporal difference
    $\hat{f} \leftarrow \hat{f} + \eta\,\delta$ # update expected reward
    $\mathbf{z}_V \leftarrow \lambda_V\,\mathbf{z}_V + \nabla_V \mathbf{V}(s)$ # update value function eligibility traces
    $\mathbf{z}_W \leftarrow \lambda_W\,\mathbf{z}_W + \nabla_W \ln \pi(a|s,\ \mathbf{W})$ # update preferences table eligibility traces
    $\mathbf{V} \leftarrow \mathbf{V} + \eta_V\,\delta\,\mathbf{z}_V$ # update value function estimate
    $\mathbf{W} \leftarrow \mathbf{W} + \eta_W\,\delta\,\mathbf{z}_W$ # update preferences table estimate
    $\mathbf{W} \leftarrow \text{sign}(\mathbf{W}) \times \min(|\mathbf{W}|,\ W_{max})$ # Clip to ensure minimal exploration rate (eq. (7.83)).
    $s \leftarrow s'$ # step forward

Listing 7.2: Pseudocode for input signature change accommodation in traditional Actor-Critic algorithm with eligibility traces, similarly to Listing 7.1. The gradient computations ∇_V and ∇_W are detailed in eqs. (7.78-7.80). The additional clipping operation described in eq. (7.83) is also highlighted.

always at least a chance p_{min} to pick a non-preferred action:

$$p_{min} = \frac{e^{-W_{max}}}{e^{W_{max}} + e^{-W_{max}}} \tag{7.84}$$

For this reason, the set of policies actually visited during our AC search is the one represented in Figure 7.2.d.

To properly compare QL and AC approaches in the experiment, the values of ε and W_{max} are setup so that the base exploration level p_{min} is the same in both methods.

Like QL, AC is unequipped to face $(+i)$ and $(-i)$ signature changes, because the associated tabular function have fixed signature. To accommodate IPL, the same transfer projection functions (see (7.70) and (7.72)) are applied to W, $\hat{V}$, z_W and z_V during the events, so the number of lines in these two tables adjust the agent new required signature. Scalar values $\hat{f}$ and δ undergo no particular adjustment. The purpose is to verify that such a low-level adaption of AC to IPL works in practice, at least on the benchmark environments.

7.3 Measures

In the previous sections, we have described that the experiment operates according to the following protocol:

1. Define the parsimonious class of deterministic, tabular RL environments as a benchmark $\mathscr{T}$.
2. Filter out irrelevant environments from the benchmark to constitute $\mathscr{T}_{\text{ssi}}$.
3. Analyze every environment in the benchmark to find the theoretical optimal policies within them: $\bar{\mathscr{P}}^*_{det}$, $\mathscr{P}^*_{det}$, $\bar{\mathscr{P}}^*_{sto}$, and $\mathscr{P}^*_{sto}$.
4. Use this information to classify the benchmark into 8 different IPL joint profiles (see Table 7.1).
5. Pick 100 uniformly random environments from each joint profile.
6. Within every environment picked:
   (a) Initialize two 1D learners (QL and AC) and two 2D learners (QL and AC) constituting the first form protean agents A_{p_1}.
   (b) Run their learning algorithm for 50 000 iterations.
   (c) Introduce $(+i)$ change event to 1D learners and $(-i)$ to 2D learners.
   (d) Adjust the learning methods with the low-level transfer projections described in eqs. (7.70) and (7.72). The corresponding agents constitute the second-form protean agents A_{p_2}.
   (e) Resume the learning algorithms for 50 000 iterations again.
   (f) Run one baseline naive learner A_n for each tested agent, directly initialized with the final signature, for 50 000 iterations.

Once all the corresponding learning trajectories have been collected, the following 3 measures are taken according to the principles described in Figure 5.3. Every measure involves means over the binary feedbacks received during the simulation. These means are comprised between 0 and the maximum possible reward F^*_T, which depends on the environment at hand. To compare the measures across environments, they are all normalized by F^*_T.

To estimate the Long-Term effect of transfer (LT), we calculate the mean difference in reward between the protean and naive agent trajectories:

$$\text{LT} = \frac{1}{F^*_T} \cdot \frac{1}{50\,000} \sum_{t=50\,001}^{100\,000} \Big(f_{A_{p_2}}(t) - f_{A_n}(t) \Big) \tag{7.85}$$

$$\text{LT} \in [-1,\ 1] \tag{7.86}$$

Positive LT measures mean that the protean agent has an advantage over the naive agent on the long run, while negative value mean the opposite so there is a negative transfer.

To estimate the immediate effect of transfer (IT), we calculate the mean difference between the first 1 000 rewards received by A_{p_2} and the last 1 000 rewards received by A_{p_1}.

This constitutes 2% of their total learning time:

$$\mathrm{IT} = \frac{1}{F_T^*} \cdot \frac{1}{1\,000} \left(\sum_{t=4\,001}^{5\,000} f_{A_{p_1}}(t) - \sum_{t=5\,001}^{6\,000} f_{A_{p_2}}(t) \right) \tag{7.87}$$

$$\mathrm{IT} \in [-1,\ 1]$$

Similarly, positive IT values mean that the change event is beneficial on the short term, while negative values indicate immediate negative transfer.

Finally, to address whether the agents eventually solved the task at hand, the last 1 000 rewards received by A_{p_2} are averaged into a measure called Last Performance (LP):

$$\mathrm{LP} = \frac{1}{F_T^*} \cdot \frac{1}{1\,000} \sum_{t=9\,001}^{10\,000} f_{A_{p_2}}(t) \tag{7.88}$$

$$\mathrm{LP} \in [0,\ 1] \tag{7.89}$$

Highest values of LP indicate that the protean agent has succeeded in following an optimal policy and sticking to it by the end of the run, while low values of LP indicate task failure. We consider the task to be solved when LP is above 80%.

The benchmark $\mathscr{T}$, and especially $\mathscr{T}_{\mathrm{ssi}}$, has been enumerated and analyzed with the Python language [Van Rossum and Drake 2009]. The formulae for $F_T(P)$ and optimal values P^* have been derived with the help of the formal analysis software Mathematica [Wolfram Research 2018]. The learning runs have been simulated and measured with the Rust language [Nichols and Safari 2019; Jung *et al.* 2021], and statistical analysis have been performed with R [R Core Team 2020].

7.4 Results

The results of the experiment conducted in this chapter are summarized on Figure 7.3. As a general trend, like with the OSL results in Chapter 6, the LT measures are positive on average, supporting that the natural and almost-natural IPL projections described in sections 7.2.3 and 7.2.4 are useful to reuse knowledge in a variety of IPL situations. From a more general perspective, they suggest that it is possible to accommodate signature changes in RL with a PL approach without particular insights into the topology of $\mathscr{P}$, and still get better results than the naive agents.

Like in the previous chapter, the advantage of PL needs to be qualified depending on the situation, for it remains weak or negative on occasions. In the next, the various actual effects in play are discussed in contrast with the expected trends. A linear model was fitted on the data within tested joint profile to address the relevance of observed varia-

tions. All interactions were considered between three experimental conditions: the change event $\{(+i),\ (-i)\}$, the learning method $\{\text{QL},\ \text{AC}\}$, and the minimum exploration rate p_{min} $\{0.01,\ 0.05,\ 0.1\}$, treated as categorical values. Each model had 59 988 degrees of freedom, and their residual standard error is summarized in Annex Table A.2.

Consistently with the numerous experimental conditions addressed, there are numerous effects in play carefully dissected in the next section. We only consider effects that rely on high-significance contrasts with $p\text{-}value \leqslant 0.001$. Similarly to the previous chapter, every effect is associated to a letter like **(A)**, **(B)**, **(C)**, *etc.* To illustrate them, a few key example runs have been extracted from the data set and drawn in Figure 7.4. They are referred to with numbers like ⑴, ⑵, ⑶, *etc.* An extended visualization is also available in Annex Figures A.1 and A.2, along with the corresponding tested environments in Figure A.3 and one extra example run in Figure A.4.

(A) *No learning takes place in flat landscapes.* First of all, under the FFF-FFF condition, PL agents A_{p_2} clearly have no long-term or short-term advantage or disadvantage against the naive agents A_n (see run ⑴). This is expected since all corresponding search landscapes are flat, so there is nothing to learn in the tested environments. Regardless of the experimental conditions, no agent can improve, because they are already optimal. This null result constitutes and validates our experimental baseline.

7.4.1 Last Performance Measures

The effects summarized in this section can be read on the LP measures (contours color in Figure 7.3). We use this value to address whether the final agent A_{p_2} has successfully solved the task.

(B) *The task at hand is solved.* Throughout the various experimental conditions, LP measures are mostly distributed above 80%. This suggests that A_{p_2} eventually succeeds in solving the tasks at hand. Note that exploration mitigates this result since the LP values diminish when p_{min} increases. This is due to the tested learning methods never "cooling" the exploration rate down. As there is always exploration going on, the optimal policies cannot be exactly followed, thus the small offset between theoretical optimal gain and actual mean reward. The higher p_{min}, the wider this offset (compare for instance ⑵ and ⑸).

(C) *The agent fails when crucial input is deleted.* There is one obvious exception to **(B)** in every $(\text{HT},\ -i)$ condition. These conditions are $(\text{FHH-FHH}\ ,-i)$, $(\text{FHH-HHH}\ ,-i)$ and $(\text{HHH-HHH}\ ,-i)$. In HT environments, the diminished search space $\bar{\mathscr{P}}$ only contains suboptimal policies compared to $\mathscr{P}$. As a consequence, it is expected that the final performance be low because A_{p_2} is diminished. This observation supports our abstract model of a learning task: the task is correctly failed if the agent has not enough information to solve it, while it is correctly fulfilled in the other situations (see run ⑶).

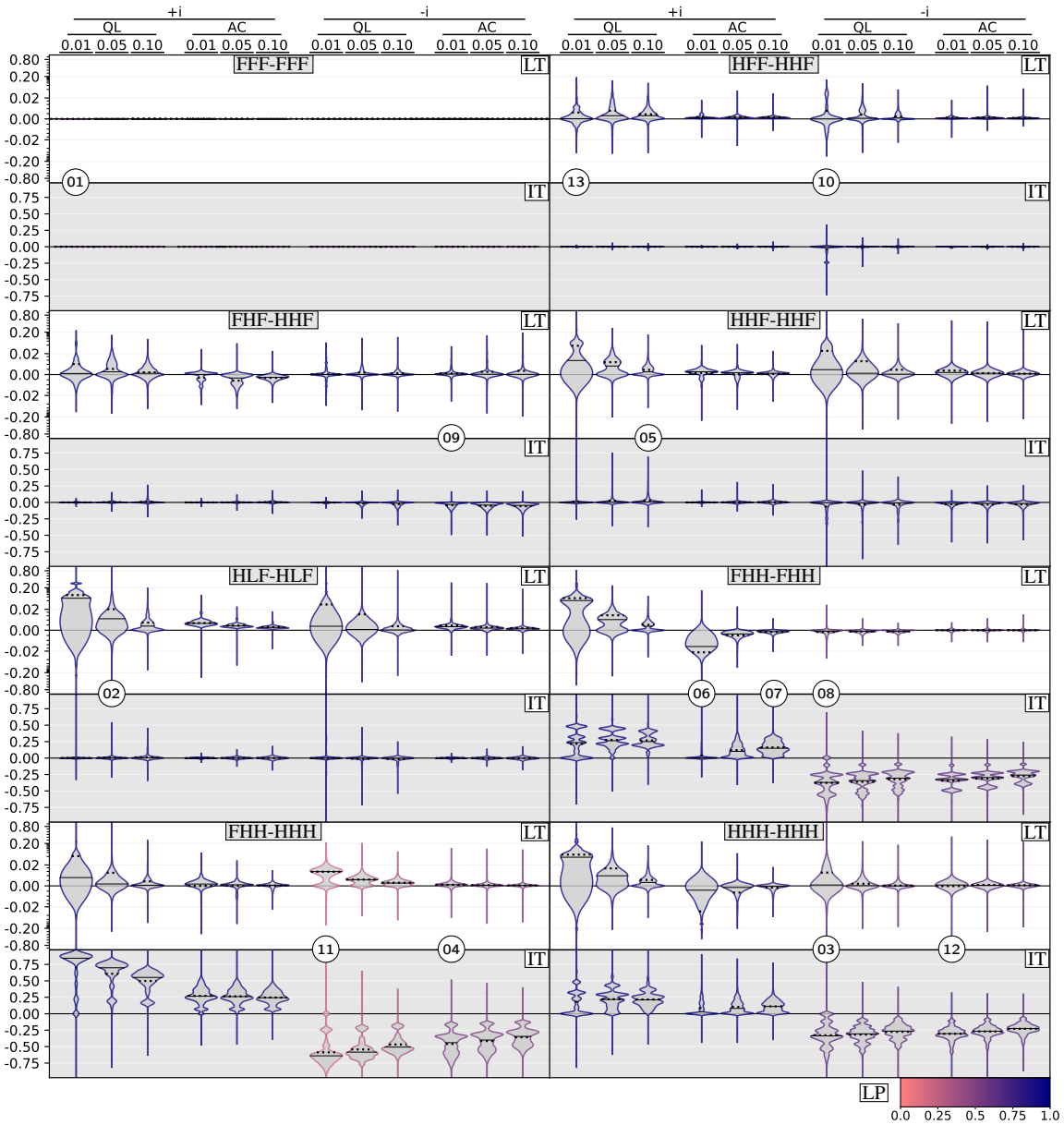


Figure 7.3: Violin plots representing the measures gathered under the various experimental conditions. Each violin summarizes 100 environments tested with 1 000 runs each. Dotted and solid lines within violins represent mean and median values. Note that LT axis is distorted to emphasize small magnitudes. Numbered labels like locate the conditions of example runs shown in Figure 7.4.

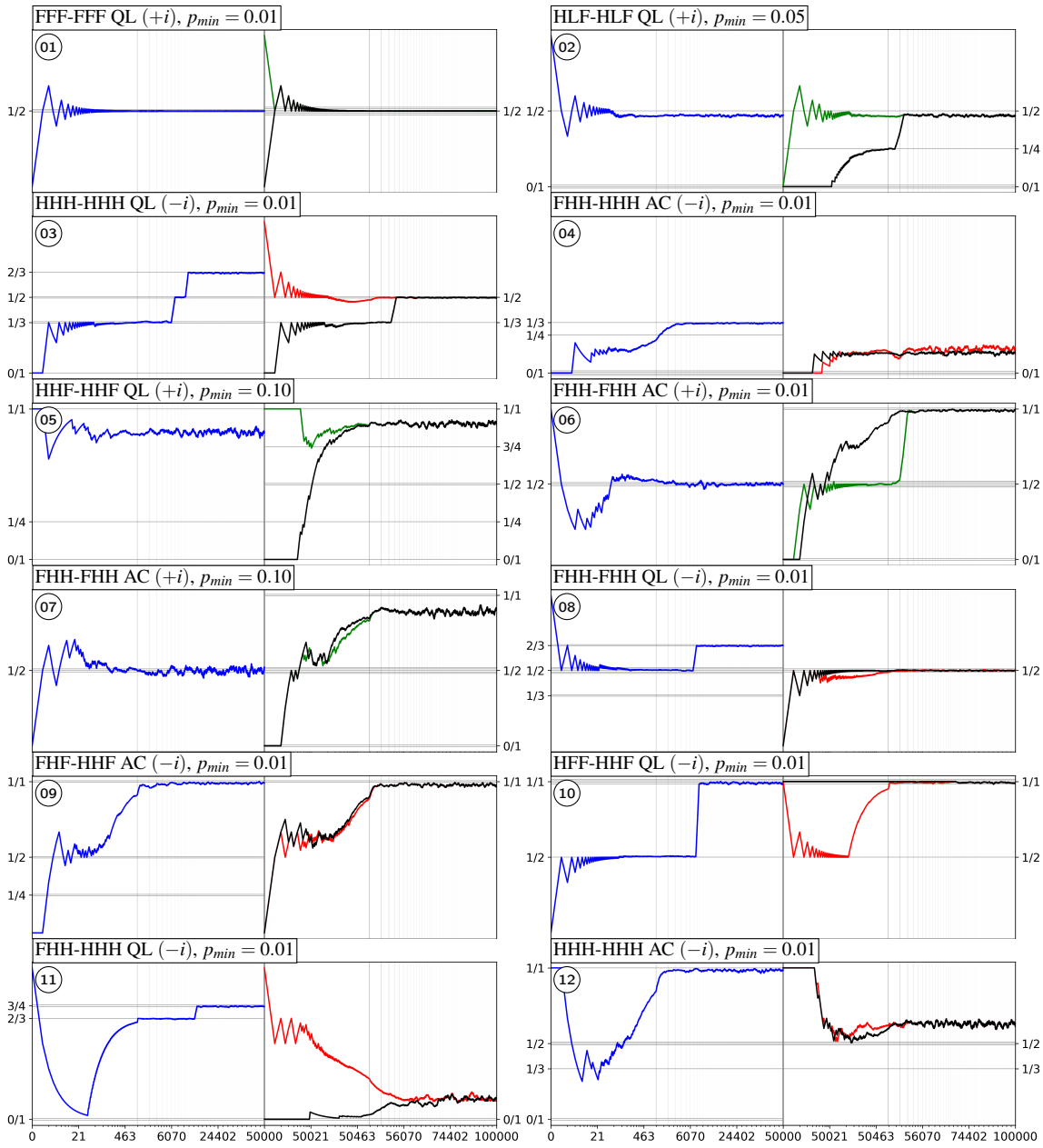


Figure 7.4: Example single runs illustrating the effects observed in Figure 7.3. The trajectories represent the binary rewards collected by the agent as a windowed average (width: 1 000 steps). For the first few steps before a window is filled, the cumulative mean is used instead. Blue trajectories (on the left of plots) correspond to A_{p_1} and continue into A_{p_2} colored trajectories (light, on the right of plots). The latter are paired with black A_n trajectories which start after the signature change event (thick vertical line). Horizontal lines mark the theoretical value of expected rewards for every deterministic cycle within current agent reach in the given environment. Note that the lines are multiplied when several cycles have the same value. Horizontal axis is distorted to emphasize the first few steps, as reminded by the thin vertical lines. The measures associated with each run can be read in Table 7.2, and the corresponding environment topologies are represented in Annex in Figure A.3, along with an extra run in Figure A.4.

run	LT.e^{-2}	IT	LP	run	LT.e^{-2}	IT	LP
⑴	0.000	0.000	1.000	⑺	-0.030	0.368	0.896
⑵	0.012	-0.006	0.926	⑻	-0.024	-0.259	0.748
⑶	0.002	0.001	0.750	⑼	0.042	-0.156	0.977
⑷	0.104	-0.501	0.429	⑽	0.070	-0.108	0.997
⑸	0.318	0.062	0.902	⑾	2.232	-0.611	0.224
⑹	-0.576	0.141	0.979	⑿	0.066	-0.367	0.630

Table 7.2: Measures values corresponding to the example runs shown in Figure 7.4.

(D) *Stochastic agents better accommodate diminished situations.* In the FHH-HHH condition, which is HT like in **(C)**, it is also observed that AC approach yields better final performance than QL. This is explained by the property (7.57). For diminished agents, the space of stochastic policies $\bar{\mathscr{P}}_{sto}$ (searched by AC) possibly contains better policies than the space of deterministic policies $\bar{\mathscr{P}}_{det}$ (searched by QL). Indeed, the 100 environments chosen with this joint profile have an average $\bar{F}^*_{det}$ value of 0.105 (std: 0.205) but an average $\bar{F}^*_{sto}$ value of 0.309 (std: 0.192). As a result, the actual mean reward collected by the agent is possibly better than the maximum possible deterministic gain (see run ⑷). This explains why this joint profile is deterministic-FI but stochastic-HI.

The effect **(D)** supports our previous claim that stochastic policies better accommodate the artefactual environmental hidden states appearing in diminished IPL situations. We still expect that recurrent policies from $\mathscr{P}_{rec}$ (not tested in this experiment) perform even better, as they infer hidden states to achieve same performances as augmented agents.

7.4.2 Long-Term Advantage Measures

The effects summarized in this section can be read on the LT measures (white panels in Figure 7.3). We use this value to assess the overall performance advantage of the PL agent A_{p_2}, that benefits from the transfer after the signature change event, against the naive agent A_n.

(E) *Natural IPL projections yield positive transfer.* As a general trend, LT measures are positive throughout the various experimental conditions, as we expect from the learning transfer that A_{p_2} benefits from. This supports that PL agents are advantaged against naive agents. As the advantage reduces when A_n catches up against A_{p_2}, A_n and A_{p_2} typically terminate the run with similar performances (see run ⑵). As a consequence, the absolute magnitude of LT values depends both on the advantage of A_{p_2} and on the value of $t_{\max}$. The higher $t_{\max}$, the longer the period of similarity and the lower the magnitude of LT as per formula (7.85). However, $t_{\max} = 10\,000$ is constant throughout the experiment, so the values can be compared to address only

the advantage, regardless of their absolute magnitude (typically below 0.2 or 0.02). A positive value of LT always indicates a long-term advantage of A_{p_2} over A_n, including either a jumpstart effect or other forms of positive transfer.

7.4.2.1 Input Addition

The effects summarized in this section can be read under the $(+i)$ conditions (leftmost groups of columns in Figure 7.3). Under these conditions, A_{p_1} is a diminished, 1D agent while A_{p_2} and A_n are augmented 2D agents.

(F) *Exploration tempers IPL advantage.* Under conditions $(\text{QL}, +i)$, we observe a general trend that LT significantly reduces as the minimal exploration rate p_{min} increases. This suggests that the naive agent A_n benefits from exploration as it catches up faster against A_{p_2} (compare for instance ⑫ and ⑮). Note that the advantage of A_{p_2} remains, as there is always $\text{LT} > 0$ on average, even with the highest tested exploration rate $p_{min} = 0.10$.

(G) *Exploration is irrelevant in easy tasks.* One exception to **(F)** is that it does not occur in the HFF-HHF condition. Every HFF landscape is both FO and FT. Also, $\left|\check{\mathscr{P}}_{det}\right|/|\mathscr{P}_{det}| = 12/16 = 75\%$ in our benchmark. As a result, in this condition, optimal policies constitute more than 75% of the deterministic search space after the $(+i)$ event. Consequently, they are likely to be discovered early even with small minimal exploration rates like $p_{min} = 0.01$ (see run ⑬ in Figure A.4 of Annex). This makes it possible for A_n to catch up early, leaving the p_{min}-independent jumpstart effect as the only benefit of A_{p_2}.

(H) *The IPL advantage is weaker with the more sophisticated learning method.* In the $(+i)$ conditions, LT scores of the AC method are consistently lower than LT scores of the QL method. One possible reason is that the continuous nature of stochastic search spaces like $\mathscr{P}_{sto}$ and $\bar{\mathscr{P}}_{sto}$, and the gradient-based nature of the AC method, make the search more efficient for both A_{p_2} and A_n, reducing the advantage of PL agents over naive agents. Another possible reason is that the preferences system W of AC (see Section 7.2.4), and the eligibility traces supposed to strengthen the direction of the search, actually make AC searches less flexible than QL when it comes to accommodating signature changes. In other terms, there would be more to *unlearn* for the AC A_{p_2} agent after the change event, which constitutes a form of negative transfer and reduces the advantage of PL. The latter hypothesis is supported by another observation, that LT is even negative in FHH-FHH and HHH-HHH situations (see run ⑥).

(I) *Exploration alleviates negative transfer.* Supporting our interpretation of **(H)**, the magnitude of negative LT measures decreases as p_{min} increases. This is consistent with the idea that exploration alleviates the effect of negative transfer, as it makes it

easier for A_{p_2} to escape local diminished optima, but still without making it better than A_n (compare runs ⑥ and ⑦). Notwithstanding the reasons for these observations, the existence of low and negative LT values in AC confirms one open hypothesis formulated in the previous chapter: the advantage of PL against naive agents is subordinate to both the learning environment and the learning method.

7.4.2.2 Input Deletion

The effects summarized in this section can be read under the $(-i)$ conditions (rightmost groups of columns in Figure 7.3). Under these conditions, A_{p_1} is an augmented 2D agent while A_{p_2} and A_n are diminished 1D agents.

(J) *The IPL advantage is weaker on input deletion.* Regarding $(-i)$ conditions, we observe a general trend that the magnitude of LT measures is an order below the LT measures in $(+i)$. One possible reason for this is that diminished policies spaces are smaller, and thus easier to explore for both A_{p_2} and A_n. This makes the advantage of PL less decisive (see run ④). For instance, in every FT environment like HLF-HLF or HHF-HHF the augmented policies space is actually not essential to achieving optimal results. It is therefore possible that A_{p_2} "recovery" after $(-i)$ is only slightly more efficient than A_n "discovery" after $(+i)$. In such a situation, the only advantage of PL happens when A_{p_1} has already converged towards an optimal diminished policy, so it is not affected at all by the event because it benefits from the almost-natural nature of the transfer projection.

(K) *No learning takes place in landscapes diminished to flat.* There is one obvious exception to **(J)** in the FHH-FHH condition. This condition is FI, so the diminished position offers no learning challenge. As a result, most LT scores are null (see run ⑧). For the same reason, the LT scores for QL methods under the FHF-HHF condition are null because FHF is FI (see run ⑨). This is consistent with the baseline constituted by the FFF-FFF condition (but see **(M)**).

(L) *The optimal degenerated policy is not always found.* In the HFF-HHF conditions, the LT measures for QL method are low. In HFF, the only possible advantage of A_{p_2} happens when A_{p_1} has already converged towards a policy within $\bar{\mathscr{P}}^*_{det}$. This unlikely because 75% of A_{p_1} search space is already occupied by optimal policies of $\check{\mathscr{P}}^*_{det}$, which is consistent with the observed effect.

(M) *There is an advantage in convergence failure.* Surprisingly, the LT measures for QL method in FHH-HHH are better than the corresponding measures in FHH-FHH, although both associated deterministic search spaces fall under the same IPL profile: FHH. This is inconsistent with **(K)**. One explanation is that QL method explores the deterministic search space while actually testing *stochastic* policies $P_{sto}\big(\hat{Q}\big)$ (see

Figure 7.2.c). This is the strategy of QL to handle the exploration *vs.* exploitation dilemma. As a consequence under this condition, A_{p_2} agents possibly benefit from property (7.57) that $\bar{F}_{sto}$ is possibly better than $\bar{F}_{det}$, like in **(D)**, although they are not supposed to (see run ⑪). This also explains that the LT measures of QL in FHH-HHH resemble the corresponding measures in HHH-HHH. In other terms, **(M)** is a possible artefact of the "off-policy" nature of QL. Closer examination of the search spaces trajectories ⑪, (see Annex Figure A.2), reveals another possible phenomenon: while the deterministic agent keeps hesitating between two deterministic policies, it actually follows a stochastic trajectory that yields better rewards than either. This is not how QL is intended to work, as it constitutes a convergence failure, resulting in artefact **(M)**.

(N) *The advantage depends on the topology exploited by the learning method.* In the HHH-HHH condition, the LT measures are positive on average with the QL method, but not with the AC method (see run ⑫). The HHH-HHH condition is the most general landscape profile and it has no degenerated "F-lat" descriptor. We suppose that this effect is due to the topology of $\mathscr{P}_{det}$ being more easy to exploit by QL than the topology of $\mathscr{P}_{sto}$ is easy to exploit by AC. This is elaborated upon in the discussion.

7.4.3 Immediate Transfer Measures

The effects summarized in this last section can be read on the IT measures (grey panels in Figure 7.3). We use this value to assess the immediate reaction of the PL agent to the signature change event.

(O) *Input changes trigger consistent immediate response.* As a general trend under HT conditions, the sign of IT measures follows the change event: positive after $(+i)$ and negative after $(-i)$. This is one expected outcome of the signature change. IPL agents are able to immediately improve when given a new source of information, but are subject to immediate regression when deprived a valuable input channel (see run ⑧). Note that this is a short-term effect so it is not redundant with **(E)** and **(J)**.

(P) *Input changes are indifferent when useless.* Unlike **(O)**, which occurs under HT conditions, IT measures are null under FT conditions. In these situations, the augmented search space does not increase the optimum, nor does the diminished search space decreases it. As a consequence, the agents react indifferently to the change event (see run ⑤). This is inconsistent with results of the previous chapter, where the IT measures were significantly negative when uninformative information channels were added during $(+i)$ events. One possible explanation to this is that the effect does occur but that our IT measure is too coarse to detect it. To verify this, we checked a second model on the same data, with finer grained IT measures where

the means rewards were computed over 100 trajectory steps instead of 1 000. No significant negative IT scores were revealed with the finer-grained measures either, which rules out this possible explanation. This is discussed in the next section.

(Q) *Exploration alleviates input deletion perturbation.* Refining **(O)**, the magnitude of negative IT measures in (HT, $-i$) conditions decreases as the minimal exploration rate p_{min} increases. This suggests that exploration alleviates the perturbation engendered by $(-i)$ by enabling faster recovery towards $\bar{\mathscr{P}}^*$ (see run ⑬).

(R) *Exploration alleviates input addition perturbation.* Regarding $(+i)$, under FHH-FHH and HHH-HHH conditions, the average IT measure for AC agents improves as the minimal exploration rate p_{min} increases. This is not observed with QL method, which is consistent with **(H)**. Both effects suggest that the AC agent A_{p_1} remains stuck within $\bar{\mathscr{P}}^*_{sto}$ longer than necessary. **(R)** suggests that exploration helps in escaping the local optimum, but **(H)** concludes that the delay lost to A_n is never overcome (see run ⑦). Interestingly, this phenomenon is not accurately observed in FHH-HHH environments, although they are also HT. We have not been able to explain the reason for this. Possibly this is due to artefactual topological specificities of the benchmark that we have not identified yet.

(S) *Response to IPL events are diverse.* Unlike previous effects, **(S)** does not rely on high-significance contrasts with $p\text{-}value \leqslant 0.001$, but on the combined observation that the distribution of IT measures under HT conditions with QL agents undergoing $(+i)$ is multimodal (see corresponding violins in Figure 7.3), and that the modes tend to collapse towards each other as p_{min} increases. In other terms, while the average IT measures varies not, two opposite trends likely interact within this condition. We suppose that after $(+i)$, some agents find the new optimum $\mathscr{P}^*$ early and struggle to stick to it as the minimum exploration rate increases. On the other hand, others struggle to find $\mathscr{P}^*$ and higher exploration rates helps them improve their search. This phenomenon reflects that the reaction to the signature change strongly depends on the environment and/or particular runs, even within one tight condition like (QL, HT, $+i$). Furthermore, **(S)** is not observed with AC agents, suggesting again that the reaction to the signature change is also strongly method-dependent.

7.5 Discussion

The experiment described in this chapter was designed to extend the results of the previous chapter, obtained in an OSL learning context, into a RL learning context (see Figure 2.2). We have chosen restricting to only tabular RL situations with small deterministic environments constituted with 4 possible states and 2 possible actions (see examples in Figure 7.1). Section 7.1.7, has shown that the corresponding benchmark $\mathscr{T}_{\text{ssi}}$ is sufficient to address a variety of IPL situations.

General Consistence The observed effect **(M)**, and the fact that **(R)** does not occur in the FHH-HHH condition, point out a few possible artefacts introduced by this particular benchmark and the chosen experimental conditions. As a consequence, the overall protocol is still subject to possible ameliorations. This said, all the observed effects **(A)**, **(B)**, **(C)**, **(D)**, **(E)**, **(K)**, **(L)** and **(O)** constitute a strong validation of our expected experimental trends and baselines: the agent tested are correctly learning, solving the task whenever possible, and failing otherwise. In particular, **(E)** supports our main result that the natural and almost-natural transfer projections described in Section 7.2 succeed in making IPL agents more efficient than naive ones. This validates our approach and suggests that PL be investigated more in depth as a promising framework to accommodating signature changes in online learning situations. **(C)** and **(O)** mitigate the above by reminding that no simple projection can restore the loss of a critical source of information after an input deletion event $(-i)$, this is further discussed in the next, along with alternate approaches to IPL.

Reaction to Useless Input Addition In FT search spaces profiles, we observe effect **(P)**. These are situations where the diminished agent has enough information to achieve the same performance as the augmented one. As a consequence, the $(+i)$ event constitutes an addition of a useless information channel, in the sense that the agent cannot use it to achieve better performance. In the experiment presented in Chapter 6, this condition triggered transient negative transfer perturbations, because it took time for A_{p_2} to figure out that the new channel was redundant or uninformative. This was measured with significantly negative IT scores, which are not reproduced in this experiment.

We suppose that the reason is that the QL and AC methods used here are more simple and predictable than the RNN method used in the previous chapter, as they use less parameters. Correspondingly, the policy landscape explored by QL and AC is smaller in terms of dimensionality, and presumably less bumpy than the high-dimensional landscapes explored by the RNN, whose sensitivity to inner parameters is very high on occasions. We suppose that this difference in parameter sensitivity explains the different results obtained here, although this remains to be rigorously measured. In any case, this particular discrepancy between both experiments strongly suggests that the learning method at hand influences even the IPL reactions to change events, even though the transfer projections are natural in both cases.

Exploration, Topology and the IPL Projections In accordance with the above, effects **(F)**, **(G)**, **(I)**, **(J)** and **(L)** demonstrate a deep connection between the IPL problem and the "exploration *vs.* exploitation" dilemma. The problem of smoothly accommodating a signature change essentially boils down to finding a new acceptable optimal policy $\hat{P}$ in a search space that has been modified, either by a $(+i)$ or a $(-i)$ structural change.

This dilemma is vastly discussed in the literature, and it is not the goal of the experiment to address it again. Instead, we address that the particular modification of $\mathscr{P}$

induced by the IPL events is unusual, and deserves consideration. With these IPL events, the modification consists in a projection into either a higher dimensional (after $(+i)$) or a lower dimensional (after $(-i)$) alternate space (see Section 5.3). Once projected, agents with a tendency to "exploit" more easily benefit from the PL transfer in situations where the projection lands them on a promising region of the new search space. On the other hand, agents with a tendency to "explore" more easily escape the trap of negative transfer in situations where the projection lands them in strongly suboptimal regions of the new search space.

To illustrate this, take for instance the effect **(N)** (run ⑫). When undergoing $(-i)$ in the HHH-HHH condition, there are various possible ways the A_{p_2} agent benefits from a PL transfer. Here are three examples:

1. A_{p_1} got mistakenly stuck on the local optimum $\bar{\mathscr{P}}^*$ while exploring $\mathscr{P}$. This constitutes a search failure, but has the fortunate consequence that A_{p_2} already stands on the only remaining optimum after $(-i)$. The chances of this happening depend both on the exploration *vs.* exploitation tradeoff struck by the learning method (*i.e.* likely to stop when reaching local optima), and on the chance inherent to the heuristic search (*i.e.* encounters $\bar{\mathscr{P}}^*$ in a stage of likely convergence).

2. $\bar{\mathscr{P}}^*$ is "close" from $\mathscr{P}^*$ with respect to a topology that the projection method is conserving. As a consequence, when projected into the diminished policies space, A_{p_2} only needs a short local exploration to find the diminished optimum again, whereas A_n has to explore all $\bar{\mathscr{P}}$ over from scratch.

3. A_{p_1} remembers having explored $\bar{\mathscr{P}}^*$ during its search over $\mathscr{P}$, and the IPL projection makes this memory available to A_{p_2}. This implies that the learning method be non-Markovian, and that the transfer technique preserves its memory. In this case, A_{p_2} directly moves back on $\bar{\mathscr{P}}^*$ after $(-i)$ without needing to explore $\bar{\mathscr{P}}$ again.

Considering that, according to effect **(B)**, the measured LP scores are typically close from their maximum possible value, we consider that all agents A_{p_1}, A_{p_2} and A_n eventually reach the true optimal policy in their current available search space. This is made possible by the chosen long term $t_{\max} = 10\,000$, and rules out the possibility 1. Since neither QL nor AC method features an explicit memory of far passed local optima encountered in the past, there are also very few chances that 3 occurs in the experiment.

As a consequence, we assume that in our benchmark, $\mathscr{P}_{det}$ exhibits an interesting topology that is both exploited by QL and conserved by the chosen almost-natural projection, but that the situation is different with $\mathscr{P}_{sto}$ and AC, thus the effect **(N)**. In conclusion, we have exhibited one example IPL projection that conserves the topology exploited by the learning method, and one that does not, suggesting that the reactions of learning agents to IPL events strongly depends on both the learning method and the transfer projection technique.

The Role of IPL The above observations illustrate two distinct responsibilities regarding heuristic learning agents design. On the one hand, the balance between "exploration" and "exploitation" tendencies is supposed to be struck by the learning method at hand, *e.g.*, QL or AC in our situation. For instance: **(Q)** suggests that a tendency to explore alleviates the perturbation caused by $(-i)$. This is outside the scope of IPL. On the other hand, it is IPL approaches' responsibility to ensure that the landing region after $(+i)$ and $(-i)$ is appropriate. *In fine*, the landing region is entirely determined by the chosen projection method, supposed to make the agent signature compatible with the new input space.

In our case, the projection is either a natural or an almost-natural projection, boiling down to simple operations on the columns of the internal agent parameters tables (see Section 7.2 for this chapter, and Section 6.6 in the previous one). These projections are simple because they do not require particular insights about the topology of $\mathscr{P}$. But the major result of the two experiment, is that they are sufficient to provide IPL agents a significant advantage against naive agents in a variety of learning contexts.

This said, the observed effects **(H)** and **(R)** confirm that negative transfer does occur when the landing region is not adequate (compare for example the policy trajectories ⑥ and ⑦ in Annex Figures A.1 and A.2). In this case, only exploration helps, but then there is no particular advantage of IPL against naive agents.

In summary, since local heuristic searches over the policies spaces $\mathscr{P}$ typically rely on a particular topology of $\mathscr{P}$, PL is responsible for accommodating input signature changes by constructing projections that make the best use of this very topology. As a result, the best IPL projection method strongly *depends* on the topology exploited by the learning method at hand.

Choosing the Policy Type From the agent perspective, the most challenging aspect of diminished environments is that they feature artefactual hidden states, as per the transition projections described in Figure 7.1. From the benchmark analysis and the experimental results, it appears that agents able to follow stochastic policies perform better than deterministic agents in this situation, because they approximate augmented trajectories that would otherwise be impossible to follow (see equation (7.57)). However, there exists alternate behavioural search spaces beyond $\bar{\mathscr{P}}_{det}$ and $\bar{\mathscr{P}}_{sto}$, containing policies able to follow the augmented trajectories exactly, and fitting none of the IPL profiles listed in Figure 5.2.

For instance, it is possible for diminished policies able to remember the last few iterations to discriminate between states a^+ and a^- even though they only receive a as an input, because the transitions leading to a^+ and a^- possibly have different antecedents. These policies need to rely on *recurrent* procedures, and feature a *memory* of past events. They constitute $\bar{\mathscr{P}}_{rec}$, a wider search space, more difficult to explore than $\bar{\mathscr{P}}_{det}$ and $\bar{\mathscr{P}}_{sto}$ in general because the convergence guarantees of traditional RL approaches, relying on Markovian properties of both policies and the environment, do not hold anymore. Notwithstanding, recurrent policies have long been the most natural fit to RL, and are investigated pervasively throughout the domain [Lin and Mitchell 1993; Boots and Gordon 2010; Sil-

ver *et al.* 2016; Tallec and Ollivier 2018].

This aspect of IPL is specific to its interaction with RL, as it results from the recurrent nature of RL. In OSL context (see Chapter 6), inputs were not related to past inputs, so no policy within $\bar{\mathscr{P}}_{rec}$ could do better than the best policy in $\bar{\mathscr{P}}_{sto}$. This is not the case in this chapter, and now that the precise interplay between $\bar{\mathscr{P}}_{det}$ and $\bar{\mathscr{P}}_{sto}$ has been carefully dissected with the current experimental results, we suggest that recurrent policies be investigated in future works as the most promising behavioural search space able to accommodate IPL signature change events in RL.

Part III

Conclusion

Chapter 8

Conclusion

8.1 Summary

In this thesis, we have been interested in a particular class of computer procedures, the *learning procedures*, that are the object of ML science. Like any computer procedure, learning procedures are characterized by their input/output signature, a rigorous contract established between the procedure and its caller. Their particularity is that they target elusive procedures $P^\star$, whose signature is also rigorously known.

The Problem Statement The contract is *mandatory* as it specifies the number, the type and the meaning of the data fed into the procedure (the input signature), and of the data produced by the procedure (the output signature): without a precise signature, the contract is ill-defined and the procedure cannot be run. For this reason, it is a frequent implicit assumption in ML that the signatures are fixed throughout the learning agents lifetimes.

However, ML addresses natural learning contexts in which not only the agent is submitted to environmental fluctuations, but also its interface with the environment. This happens for instance with the roverbot described in Section 1.2, whose signature changes whenever a sensor or an actuator is added or removed from its external interface. This also happens with streaming statistical predictors, like internet agents analysing the behaviour of online users, as they feed from data streams whose relevance or availability cannot be relied upon over the course of learning. In these situations, the signature of the learning agent cannot be kept, nor dropped, so it needs to *transform* for the learning process to keep working.

The Proposition Accommodating signature changes of learning agents is the object of the learning situation introduced in this thesis, namely Protean Learning (PL). Chapters 2 and Chapter 4 have sketched a large overview of ML that positioned PL as a subdomain of Transfer Learning (TL), transversely to Supervised Learning (SL), Unsupervised Learning (UL) and an exclusive case of Online Learning (OL): Reinforcement Learning (RL). We have defended that PL constitutes a significant, new, and non-trivial domain of ML,

addressing all natural learning situations challenged by potential changes to the agent signature.

The purpose of PL is twofold. First, PL aims to construct learning agents that do not become undefined after signature changes. Second, and similarly to TL, PL aims that their performance after the change be at least better than the performance of naive agents resuming the learning from scratch on every change. In Chapter 5, we have developed, with an original formalism dedicated to describe streaming procedures, a rigorous formalization of PL as a stream processing issue. This formalization makes it explicit what is considered a "signature change" in PL, and the objective of PL.

The Contributions In Section 5.3 and subsequent chapters, we have restricted our focus on two elementary signature change events: input addition $(+i)$ and input deletion $(-i)$, constituting the restricted domain of Input-PL (IPL). In Chapter 5, we have shown that $(+i)$ and $(-i)$ reveal a non-trivial range of situations possibly met by IPL agents: a set of 10 different landscape profiles describing possible configurations of the explored search space. Regarding these, we have discussed how a precise *ad hoc* knowledge of these landscapes is useful in designing dedicated IPL algorithms accommodating the change events.

Moreover, we have unveiled a natural set of projections, namely the *natural projection* ℓ for $(+i)$ and the class of *almost-natural projections* $\not\rho$ for $(-i)$, which permit generic accommodation of these events regardless of the task at hand. These projections meet the first objective of PL *de facto*, because the resulting agents remain correctly defined. In this thesis, we have also defended that they meet the second objective. We defended that natural IPL projections constitute an acceptable generic way of accommodating IPL signature changes, and produce better results than naive approaches.

In Chapter 6, we verified this claim in an Online Supervised Learning (OSL) context. With a carefully controlled synthetic experiment, we showed that the natural projections could be applied under the form of structural editions to a RNN-based learner. The resulting agents learned better than naive ones on most tested tasks. With a precise analysis of the various effects in play during IPL accommodation, we showed in particular that the technique was most useful after $(-i)$ when the remaining inputs were sufficient to solve the task at hand. Also, superfluous inputs added with $(+i)$ triggered transitory negative transfer perturbations that were overcome by IPL agents on the long run. We concluded that natural IPL projections, in spite of their genericity, are at least moderate enough not to perturb the ANN structure, and efficient enough to yield better results than the naive baseline in OSL.

In Chapter 7, we extended the above verification to a full-fledged RL context. With a carefully crafted dedicated benchmark, we showed that the natural projections could also be applied to different learning algorithms like Q-Learning and Actor-Critic. Once again, the resulting agents learned better than naive ones on most tested tasks. To qualify this result, with another analysis of the various effects in play during IPL accommodation, we showed that the IPL advantage varied much depending on the learning context and

the algorithm at hand. In particular, we showed that diminishing $(-i)$ events could transform a Markovian RL environment, easily tackled with deterministic policies, into a non-Markovian environment featuring artefactual hidden states, which only recurrent policies can address exactly. As a consequence, not only the signature of the learning agent becomes inadequate after $(-i)$, but also the learning algorithm itself. We concluded that natural projections remain the best choice in the absence of *ad hoc* information about the task or future signature change events, but also that any insight into the particular topology of the explored search space should be used to improve PL performances.

In summary, we have contributed to the definition, the positioning and the study of PL. We have constructed a first tool, the *natural projections*, dedicated to tackle IPL in particular, and evaluated the value of this tool in various contexts. Natural IPL projections are advantageous because they correctly accommodate IPL events regardless of the task at hand. But they are also limited as they do not exploit particularities of the task at hand. With a pair of synthetic experiments, we have carefully listed, quantified and documented these limitations.

8.2 Limits and Perspectives

The work presented in this thesis constitutes preliminary groundwork for the domain of PL, whose foundation we intended to contribute to. As such, it is essentially incomplete, there is room for much theoretical improvement and many open questions deserve to be addressed in future works.

Regarding Theoretical PL From a theoretical perspective, the formalization offered in Chapter 5 is both generic and consistent, but has been tailored to represent recurrent PL in RL so it is not straightforward to extend to every learning situation. For instance, the diagrams needs to be reinterpreted as described in Section 5.2.7 to represent other forms of OL like Online UL or Online SL (OSL). Future extensions of the model should enable seamless use of diagrams like (5.26) to describe a wider range of learning situations.

Regarding IPL projections, we have suggested that natural projections be used as a generic way to accommodate signature changes before falling back to *ad hoc* techniques. However, it is an open question whether there exist other forms of generic projections, and whether they would yield better performance than the natural ones. In addition, there are several levels of *ad hoc* knowledge: the user either knows nothing about the landscape at hand, or they know its IPL profile, or the number of local optima, *etc.* As a consequence, there must exist several levels of projection genericity: projections generic to any HHF profile, or generic to any concave landscape *etc.* These projections would constitute a useful range of method for PL users to pick within, but still remain to be unveiled.

The results of Chapter 7 strongly suggest that the benefit of generic IPL projections depends not only on the transfer technique, but also on the learning method and the ex-

plored search space. Most notably, the IPL advantage is better with stochastic agents than deterministic ones since stochastic policies better accommodate the environmental hidden states. And recurrent policies are even more promising as they possibly infer the missing states information. In other terms: every IPL landscape is FFF-FFF to a recurrent protean agent provided it has enough memory to discriminate ambiguous inputs. As of today, there is no theory for the interplay between the learning method and the advantage of generic IPL projections. And it is unsure, in the general case, whether signature changes should be better tackled with a careful choice of transfer projection or of the explored policies space.

Regarding Practical PL From a practical perspective, the natural projections have proven to be simple and generic enough that they can be seamlessly applied in three radically different learning algorithms: an RNN-based OSL learner, vanilla Q-Learning and an Actor-Critic method enhanced with eligibility traces. However, the provided advantage has been addressed in only two restricted, synthetic situations: batch OSL and tabular RL. And the IPL advantage has only been verified against naive agents on the long run.

This leaves numerous real-world questions aside. For instance, in a situation where immediate transfer matters more than long term performances, how likely is it that the "transitory" negative transfer perturbation occasionally observed in Chapter 6 cancels the benefits of using a PL approach against the naive accommodation? Again, the best long-term scores obtained in the OSL experiment range between $\mathrm{LT} = 2$ and $\mathrm{LT} = 4$ for a 1 000 steps long learning session. This is strictly more than the baseline, but is it sufficient to address real-world OSL contexts? Is it possible to quantify the advantage of generic IPL with respect to a more meaningful practical baseline?

The assessment of IPL against a tabular RL benchmark in Chapter 7 is also limited by the parsimonious nature of the chosen environments. For instance, it is yet unsure whether the results would seamlessly hold if the $(+i)$ event transformed not only 1D agents to 2D agents, but 3D agents to 4D agents *etc.* or if numerous inputs were added/deleted at once. More generally, tabular environments are notorious for being a very restricted, theoretical subset of real-world RL situations, and the algorithmic guarantees backing the traditional RL methods do not hold outside tabular contexts [Sutton and Barto 2018].

Regarding the RL algorithms tested, QL targets improvement of the future discounted reward G (equation (7.61)), so it may not converge towards the same "optimal policies" than the ones listed in Section 7.1.7 and used as experimental baselines. We expect this limitation of the experimental design to be overcome in future works by using algorithms explicitly targeting the mean average reward instead, like R-Learning [Schwartz 1993].

In summary, practical assessment of IPL projections will greatly benefit from experimental applications to real-world problems, like modular robotics [Ababsa *et al.* 2014; Doncieux *et al.* 2015] or adaptive games [Francillette 2014; Bonnici *et al.* 2019], since they demand that more specific situations be tackled to scale. This is the object of future works.

Regarding PL in General Throughout the thesis, our positioning with respect to the overall approach to PL was twofold. On the one hand, we have introduced PL as a wide subdomain of ML, supposed to tackle every TL issue with a concern about variable signatures. On the other hand, we have restricted our focus to only IPL events $(+i)$ and $(-i)$ in the Prior Transfer **(4)** situation (*cf.* page 69), so that they were specifically, and thoroughly addressed from the theory to the various synthetic experimentations. As a consequence, large areas of PL remain unexplored.

For instance, although negative transfer was controlled with care in our experiments, it was not tested whether the natural IPL projections could efficiently avoid "Catastrophic Forgetting" (CF). This phenomenon is well known from the TL community, especially regarding Subtasking TL situations **(2)**, and especially because ANNs react badly to structural editions [J. Kirkpatrick *et al.* 2017]. In both experiments, not only we obtained satisfying results in either $(+i)$ or $(-i)$ direction, but also we showed that the natural projections are moderate enough that a RNN-based agent reacted smoothly to them. The question whether CF is also overcome by the natural projections is still open, but we expect that the protocols described in Chapters 6 and 7 will be easily adapted to address this issue.

Chapter 4 has sketched a roadmap of the various challenges induced by every different PL event: not only $(+i)$ and $(-i)$, but also the *input change* $(\sim i)$, the output events $(+o)$, $(-o)$ and $(\sim o)$, and the feedback events $(+f)$, $(-f)$ and $(\sim f)$. We have explained why these events differ wildly in the obstacles they raise, but only addressed two of them currently. In the future, we expect that these various alternate events be addressed in turn, as PL progresses towards the construction of full-fledged, autonomous protean agents.

8.3 Closing Thoughts

Artificial agents developed in the context of AI result from an entanglement of human motivations. Agents are designed to address urging needs like, for instance in ML, the need to save scarce computing resources when tackling ambitious learning tasks (time, power, memory, storage, development, money, *etc*). This *efficiency* concern conflicts with a general requirement that ML agents be also *autonomous*, especially because they are expected to tackle automation problems that humans themselves do not know how to solve. The end tools constructed on this basis hinge on both technological progress and advances in the corresponding academic fields, while the ethics of their actual usage is ultimately under the supervision of our only responsible citizenship.

Notwithstanding, we believe that the essence of AI in general is more simple. As an attempt to automate tasks handled by humans, AI remains a creative activity inspired by nature. Real-world phenomena involve sophisticated natural agents able to face unanticipated situations in adequate and original ways, including ourselves. Our understanding of autonomous procedures relies on the observation of such processes, as it helps constructing our way beyond the current limitations of the artificial agents we design. As a

consequence, it is common that contemporary solutions to ML issues are derived from general knowledge of natural sciences, like CNNs from visual cortex, to address the limits of vanilla ANNs in artificial vision, genetic algorithms from evolution, to address the limits of exact maximization approaches, and RL from developmental psychology.

But in addition to rooting AI, nature is also the source of unaddressed challenges. In this thesis, we have observed that natural agents, like tadpoles undergoing metamorphosis, but also artificial agents facing natural situations, like the modular roverbot whose interface experiences structural editions, are challenged with occasional changes to their capabilities, which we have called "signature changes". Tadpoles belong to this class of natural processes seamlessly facing the situation induced by these spontaneous changes, but artificial agents are, to the best of our knowledge, unable to undergo such radical transformations.

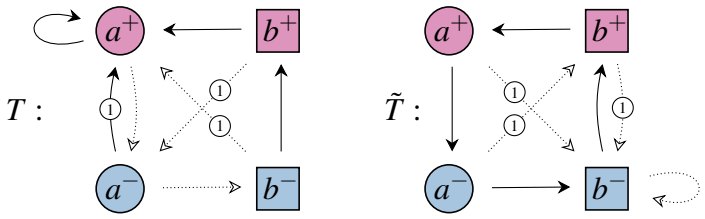
Grounded in this natural inspiration, we have uncovered a significant unaddressed issue in ML: making artificial agents *protean*, or able to accommodate changes in their very capabilities. With the natural projections offered and evaluated in our contributions, we have attempted to strike a first balance between efficiency and autonomy of such agents, so they can be used in turn to meet the various needs of ML. With the help of two scrupulous experiments addressing traditional ML contexts, we have shown that the natural projections are generic enough to be used against a variety of tasks unanticipated by the designers, and efficient enough to yield better results than the naive approaches.

Like any other ambition of ML, the pursuit of an ideal protean agent is still open, especially because the present thesis is only one contribution to it. In the future, we expect not only that improvement in this direction will benefit to all natural applications of PL like modular robotics, adaptive programs and developmental learning, but also that the flexibility provided by protean adaptation will keep paving the way towards audacious and valuable bio-inspired products of AI, and artificial life in general.

Annex

A.1 Chapter 7 Transitions Function Symmetries

Not every transition function is unique from the learner agent perspective. For instance, consider the following two environments T and $\tilde{T}$:



(A.1)

The representation of T and $\tilde{T}$ look different, but they only differ by their labels, and the learning dynamics in either would always be the same in the experiment. In other terms, among all automorphisms in $\mathcal{T}$, some are meaningless transformations $M : T \mapsto \tilde{T}$ that do not change the environment from the agent perspective. In their most generic form, relabelling automorphisms of $\mathcal{T}$ are defined as a sequence of functions transforming every label into another one, *e.g.*, in the case of the above transformation:

$$M : \left(\begin{array}{ll} M_{a^+} : \begin{cases} \bullet \mapsto \circ \\ \circ \mapsto \bullet \end{cases}, & M_{b^+} : \begin{cases} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{cases} \\ M_{a^-} : \begin{cases} \bullet \mapsto \circ \\ \circ \mapsto \bullet \end{cases}, & M_{b^-} : \begin{cases} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{cases} \end{array}, \begin{cases} a \mapsto b \\ b \mapsto a \\ + \mapsto - \\ - \mapsto + \end{cases} \right) \tag{A.2}$$

Testing both T and $\tilde{T}$ in the benchmark is redundant, so it can be reduced. To this end, we need to characterize "meaningless" transformations. For convenience, the same symbol M is used to refer to various automorphisms simultaneously defined over the set of input labels, action labels, transition functions, *etc.* *E.g.*, $M(a) = b$, $M(-) = +$, $M(T) = \tilde{T}$. States labels are accordingly modified the straightforward way, *e.g.*, in the above example: $M(a^+) = M(a)^{M(+)} = b^-$. Note that output labels are transformed *locally* within the context of each state, *e.g.*, $M_{a^+}(\circ) = \bullet$, but $M_{b^+}(\circ) = \circ$, so they are possibly changed in a state but not in another. As a consequence, the order of transformations matters, and we choose that action labels be transformed before states labels.

Also note that we only consider automorphisms of $\mathscr{T}$, so every transformation in M is bijective, and the reverse function M^{-1} with associated constitutive reverse transformations $M_{a^+}^{-1}$, $M_{a^-}^{-1}$, *etc.* always exists. A generic form of relabelling automorphisms like the one shown in (A.2) is:

$$M \in \left(\{ \mathscr{O} \leftrightarrow \mathscr{O} \}^{|\mathscr{S}|} \times \{ \mathscr{I}_1 \leftrightarrow \mathscr{I}_1 \} \times \{ \mathscr{I}_2 \leftrightarrow \mathscr{I}_2 \} \right) \tag{A.3}$$

Where $\{A \leftrightarrow B\}$ represents the set of all bijections between A and B. The corresponding bijection in $\{\mathscr{T} \leftrightarrow \mathscr{T}\}$ is defined by the following symmetrical property for every transition $T(s,\, u) = (s',\, f)$:

$$M(T)(M(s),\, M_s(u)) = \left(M\left(s'\right),\, f\right) \tag{A.4}$$

To specify what a "meaningless" transformation is, we need to consider the set of possible "reactions" of the agent to an environmental state. A reaction has the form $R : \mathscr{S} \to \mathscr{O}$. This represents, at the same time, the agent policy, its possible internal states, the current stochastic realization, *etc.* to determine what the next output is depending on latest environmental state. An automorphism M is meaningless in terms of learning dynamics if and only if, for every possible agent reaction R, there exists another symmetrical reaction $M(R)$ such that:

$$\forall s \in \mathscr{S},\, M_s(M(R)(M(s))) = R(s) \tag{A.5}$$

In other terms, in the transformed environment, an agent with the transformed reaction $M(R)$ behaves the exact same way as the original one, except that the labels have been changed. As a consequence, there is no need to test both the original and the transformed environments in the benchmark.

Reversibility of M ensures that $M(R)$ is always injectively specified by the meaningless property (A.5), *i.e.*:

$$M(R) : s \mapsto M_{M^{-1}(s)}^{-1}\left(R\left(M^{-1}(s)\right)\right) \tag{A.6}$$

In this view, automorphisms M do not only transform labels and transition functions into one another, but also constitute an automorphism over the set of possible reaction functions R.

Interestingly, while 2D agents possibly exhibit any possible reaction function $R_{2\mathrm{D}} = R$, 1D agents cannot react differently to indistinguishable environmental states and have restricted reactions $R_{1\mathrm{D}} = \bar{R}$. In fact, for any input $l \in I_1$ used as a label, there is always:

$$\bar{R}\left(l^+\right) = \bar{R}\left(l^-\right) \tag{A.7}$$

As a consequence, 1D reactions only constitute a subset of possible reaction functions. In $(+i)$ and $(-i)$ experiments, agents always undergo a 1D experience, either before or af-

ter the signature change event. This sets a constraint on the set of meaningless relabelling transformations M. To remain meaningless in this context, M must make it possible that a transformed 1D agent exhibits the same reaction as any original 1D agent. In other terms, M must also constitute an endomorphism over the restricted set of 1D reaction functions, so that $M(\bar{R})(l^+) = M(\bar{R})(l^-)$ is also always true. In the following, we demonstrate that this constraint is exactly:

$$\forall l \in \mathcal{I}_1,\ M_{l^+} = M_{l^-} \tag{A.8}$$

Which permits to rule out about 94% redundant environments from the benchmark, without reducing its diversity.

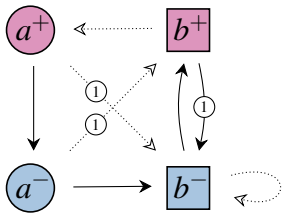
To show that the condition (A.8) is necessary, we consider a meaningless transformation M that violates it, so:

$$\exists (l,\ o) \in \mathcal{I}_1 \times \mathcal{O},\ M_{l^+}(o) \neq M_{l^-}(o) \tag{A.9}$$

Take for instance the following transformation:

$$M : \left(\begin{array}{ll} M_{a^+} : \begin{cases} \bullet \mapsto \circ \\ \circ \mapsto \bullet \end{cases}, & M_{b^+} : \begin{cases} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{cases} \\ M_{a^-} : \begin{cases} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{cases}, & M_{b^-} : \begin{cases} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{cases} \end{array}, \begin{cases} a \mapsto b \\ b \mapsto a \\ + \mapsto - \\ - \mapsto + \end{cases} \right) \tag{A.10}$$

In this relabelling, $M_{a^+}(\bullet) \neq M_{a^-}(\bullet)$, which yields the following transformed transition function $M(T)$ according to (A.4):



(A.11)

Since M is supposed to be bijective over the set of every possible 1D reactions, then no matter the diverging output o considered, there must exist a 1D reaction $\bar{R}$ whose image $M(\bar{R})$ yields exactly o when faced with environmental state $M(l^+)$:

$$\exists \bar{R},\ M(\bar{R})\big(M\big(l^+\big)\big) = o \tag{A.12}$$

In our example, this could be $M(\bar{R}) : b^- \mapsto \bullet$. Now, for every $l \in \mathcal{I}_1$:

$$\begin{aligned} M\big(l^+\big) &= M(l)^{M(+)} \\ M\big(l^-\big) &= M(l)^{M(-)} \end{aligned} \tag{A.13}$$

So the above two states $M(l^+)$ and $M(l^-)$ only differ by their $\mathscr{I}_2$ label (exponent sign). According to (A.7), 1D reactions are insensitive to $\mathscr{I}_2$ labels, so we can write:

$$M(\bar{R})\big(M\big(l^+\big)\big) = M(\bar{R})\big(M\big(l^-\big)\big) = o \tag{A.14}$$

Reinjecting these in (A.9), we get:

$$M_{l^+}\big(M(\bar{R})\big(M\big(l^+\big)\big)\big) \neq M_{l^-}\big(M(\bar{R})\big(M\big(l^-\big)\big)\big) \tag{A.15}$$

Which by the meaningless property (A.5) reduces to:

$$\bar{R}\big(l^+\big) \neq \bar{R}\big(l^-\big) \tag{A.16}$$

This contradicts with the basic property of 1D reaction functions (A.7), and M is not an endomorphism over 1D reactions anymore. Indeed, with the above transition function $M(T)$ (A.11), a 1D agent that picks ● when observing b would loop between b^+ and b^- states, with $M(\bar{R})(b^+) = M(\bar{R})(b^-)$. However, in the corresponding preimage transition function T (A.1), looping between corresponding preimage states a^- and a^+ would require that ○ be taken in a^+ and ● in a^-, so $R(a^+) \neq R(a^-)$, which is impossible for an agent which cannot distinguish between a^+ and a^-. As a consequence, no meaningless transformation M can verify (A.9), they must verify (A.8) instead.

Now, to show that (A.8) is a sufficient condition for M to be meaningless, we start from the basic property of any 1D reaction function $\bar{R}$:

$$\forall l \in \mathscr{I}_1,\ \bar{R}\big(l^+\big) = \bar{R}\big(l^-\big) \tag{A.17}$$

In particular, this is true for any preimage state $M^{-1}(l)$:

$$\bar{R}\Big(M^{-1}(l)^+\Big) = \bar{R}\Big(M^{-1}(l)^-\Big) \tag{A.18}$$

According to (A.7), $\bar{R}$ is indifferent to $\mathscr{I}_2$ labels (exponent sign), so it is safe to also write:

$$\bar{R}\big(M^{-1}\big(l^+\big)\big) = \bar{R}\big(M^{-1}\big(l^-\big)\big) \tag{A.19}$$

The inverse version of (A.8) yields that:

$$\forall l \in \mathscr{I}_1,\ M^{-1}_{l^+} = M^{-1}_{l^-} \tag{A.20}$$

In particular, and because $M^{-1}(l^+)$ and $M^{-1}(l^-)$ only differ by their $\mathscr{I}_2$ labels due to (A.13), we have:

$$M^{-1}_{M^{-1}(l^+)} = M^{-1}_{M^{-1}(l^-)} \tag{A.21}$$

Reusing this in (A.19), we get:

$$M^{-1}_{M^{-1}(l^+)}\big(\bar{R}\big(M^{-1}\big(l^+\big)\big)\big) = M^{-1}_{M^{-1}(l^-)}\big(\bar{R}\big(M^{-1}\big(l^-\big)\big)\big) \tag{A.22}$$

Which according to (A.6) reduces to:

$$M(\bar{R})\big(l^+\big) = M(\bar{R})\big(l^-\big) \tag{A.23}$$

This shows that the transformed reaction $M(\bar{R})$ also respects the property (A.7). So if M respects condition (A.8), then it is also an endomorphism over the set of 1D reactions functions, which makes this condition sufficient.

In the end, a meaningless relabelling automorphism M that correctly verifies (A.8) is completely summarized under the shorter form:

$$M : \left(M_a : \left\{ \begin{matrix} \bullet \mapsto \circ \\ \circ \mapsto \bullet \end{matrix} \right., \; M_b : \left\{ \begin{matrix} \bullet \mapsto \bullet \\ \circ \mapsto \circ \end{matrix} \right., \; \left\{ \begin{matrix} a \mapsto b \\ b \mapsto a \end{matrix} \right., \; \left\{ \begin{matrix} + \mapsto - \\ - \mapsto + \end{matrix} \right. \right) \tag{A.24}$$

With only the first input channel $\mathscr{I}_1$ used as a context to modify output labels with bijections M_a and M_b. As a consequence, the generic form of meaningless transformations M (A.3) can be rewritten:

$$M \in \Big(\big\{ \mathscr{O} \leftrightarrow \mathscr{O} \big\}^{|\mathscr{I}_1|} \times \big\{ \mathscr{I}_1 \leftrightarrow \mathscr{I}_1 \big\} \times \big\{ \mathscr{I}_2 \leftrightarrow \mathscr{I}_2 \big\} \Big) \tag{A.25}$$

So we count that there exists

$$|\mathscr{O}|!^{|\mathscr{I}_1|} \times |\mathscr{I}_1|! \times |\mathscr{I}_2|! = 2!^2 \times 2! \times 2! = 16 \tag{A.26}$$

possible meaningless transformations in the benchmark, including identity.

With these meaningless transformations correctly characterized, transition functions in $\mathscr{T}_{\text{rec}}$ are easily partitioned into C equivalence classes:

$$\begin{gathered} \mathscr{T}_{\text{rec}} = \bigcup_{c=0}^{C} \mathscr{T}_{\text{eq}}(c) \\ \forall c,\, c' \in \{1,\, \ldots,\, C\},\, c \neq c' \implies \mathscr{T}_{\text{eq}}(c) \cap \mathscr{T}_{\text{eq}}\big(c'\big) = \varnothing \\ \forall c,\, \forall T,\, \tilde{T} \in \mathscr{T}_{\text{eq}}(c),\, \exists M,\, M \text{ meaningless } \&\, M(T) = \tilde{T} \end{gathered} \tag{A.27}$$

Within an equivalence class, all environments are meaningless relabellings of each other, so they are redundant and only one of them needs to be tested in the benchmark. Note that a few classes contain less than 16 elements since the corresponding transition functions have structural symmetries, resulting in $M(T) = T$ even when M is not identity. Careful screening of $\mathscr{T}_{\text{rec}}$ yields the following count:

$$C = 340\,136 \tag{A.28}$$

For the sake of consistency and reproducibility of the benchmark, one environment has to be deterministically chosen within each symmetry class. We first decide a canonical order $\mathbb{O}$ on the binary symbols used to define the transition function:

$$\begin{aligned}\mathbb{O}(\mathscr{O}) &= (\bullet,\ \circ)\\ \mathbb{O}(\mathscr{F}) &= (0,\ 1)\\ \mathbb{O}(\mathscr{I}_1) &= (a,\ b)\\ \mathbb{O}(\mathscr{I}_2) &= (+,\ -)\\ \mathbb{O}(\mathscr{S}) &= \left(a^+,\ a^-,\ b^+,\ b^-\right)\end{aligned} \tag{A.29}$$

Then, we encode each transition function as the successive symbols needed to define it completely:

$$\forall T \in \mathscr{T},\ \mathbb{O}(T) = \sum_{(s,\ o)\in\mathbb{O}(\mathscr{S})\times\mathbb{O}(\mathscr{O})} T(s,o) \tag{A.30}$$

Where the $\times$ operation is defined as the ordered product of tuples, *e.g.*:

$$(a,\ b)\times(\bullet,\ \circ) = ((a,\ \bullet),\ (a,\ \circ),\ (b,\ \bullet),\ (b,\ \circ)), \tag{A.31}$$

And the $+$ operation is defined as the ordered concatenation of tuples, *e.g.*:

$$(x_1,\ y_1)+(x_2,\ y_2) = (x_1,\ y_1,\ x_2,\ y_2) \tag{A.32}$$

For instance, canonical encoding for environment T_A (see Figure 7.1) is:

$$\mathbb{O}(T_A) = \left(a^+,\ 0,\ b^-,\ 0,\ a^+,\ 1,\ a^+,\ 0,\ a^-,\ 0,\ b^-,\ 0,\ a^+,\ 0,\ b^+,\ 0\right) \tag{A.33}$$

This provides a strict total, lexicographic ordering of $\mathscr{T}$, and *a fortiori* a total ordering within each symmetry class $\mathscr{T}_{\text{eq}}(c)$. For instance, given the two environments defined in (A.1):

$$T < \tilde{T} \tag{A.34}$$

To constitute the benchmark, only the first element of each $\mathscr{T}_{\text{eq}}(c)$, with respect to this ordering, is kept:

$$\mathscr{T}_{\text{can}} = \left\{\min\left(\mathscr{T}_{\text{eq}}(c)\right),\ \ c\in\{1,\ \ldots,\ C\}\right\} \tag{A.35}$$

Consistently with (A.26), this reduces the benchmark by about $15/16 \approx 94\%$.

A.2 Constraints On Joint Profiles

Not every combination of IPL profiles into a joint profile is possible. In the context of our benchmark, there are a few particular constraints which we expound here. They are all summarized in Table A.1

As long as there exists one non-optimal path within the transition function (*e.g.*, the

if $\mathscr{P}_{det}$ is	then $\mathscr{P}_{sto}$ is not	
HI	FI	(A.38)
HT	LO	(A.40)
HT	FT & FI	(A.41)
FT	HT	(A.39)
LO HO	FO	(A.37)
any	FFH HFF HFH	(A.36)

Table A.1: Summary of constraints on IPL joint profiles.

transition $(a^+, \circ)$ in T_E (7.58)), then a policy that deterministically avoids it is always better than a stochastic policy with a chance of undergoing it. As a consequence, the only possible optimal stochastic policies within $\mathscr{P}_{sto}$ contain at least one degenerated deterministic component. In other terms, one associated optimal probability measure is either exactly 0 or 1, so the optimal policies must lie on the boundary of the corresponding hypercube $[0,\ 1]^{|\mathcal{S}|}$. Since there exists policies within $\check{\mathscr{P}}_{sto}$ that do not lie on this boundary, this implies that the outer landscape of F cannot be flat:

$$\mathscr{P}^*_{sto} \neq \mathscr{P}_{sto} \implies \check{\mathscr{P}}^*_{sto} \neq \check{\mathscr{P}}_{sto} \tag{A.36}$$

As a result, no stochastic landscape $\mathscr{P}_{sto}$ can belong to the FO categories HFF, FFH or HFH.

A consequence of (7.34) and (7.60) is that no environment with a deterministic landscape classified as LO or HO can exhibit a FO stochastic landscape since suboptimal deterministic policies of $\mathscr{P}_{det}$ are inherited by $\mathscr{P}_{sto}$:

$$\check{\mathscr{P}}^*_{det} \neq \check{\mathscr{P}}_{det} \implies \check{\mathscr{P}}^*_{sto} \neq \check{\mathscr{P}}_{sto} \tag{A.37}$$

The same reasoning also holds for diminished policies $\bar{\mathscr{P}}$: no environment with a deterministic HI landscape can belong to a stochastic FI category:

$$\bar{\mathscr{P}}^*_{det} \neq \bar{\mathscr{P}}_{det} \implies \bar{\mathscr{P}}^*_{sto} \neq \bar{\mathscr{P}}_{sto} \tag{A.38}$$

Regarding the best policy values, a consequence of (7.57) is that no environment with a deterministic FT landscape can be also classified as a stochastic HT:

$$\bar{F}^*_{det} = F^*_{det} \implies \bar{F}^*_{sto} = F^*_{sto} \tag{A.39}$$

And as a consequence of (7.60), no environment with a deterministic HT landscape

can be classified as a stochastic LO:

$$\bar{F}^*_{det} < F^*_{det} \implies \check{\mathscr{P}}^*_{sto} \neq \varnothing \tag{A.40}$$

When a deterministic HT landscape extends into a stochastic FT landscape, it means that some diminished stochastic policies yield better results than diminished deterministic ones, like in the example T_A. But since $\bar{\mathscr{P}}_{sto}$ inherits from all policies in $\bar{\mathscr{P}}_{det}$, including the inferior ones, then the corresponding stochastic landscape cannot be FI:

$$(\bar{F}^*_{det} < F^*_{det} \;\&\; \bar{F}^*_{sto} = F^*_{sto}) \implies \bar{\mathscr{P}}^*_{sto} \neq \bar{\mathscr{P}}_{sto} \tag{A.41}$$

A.3 Chapter 7 Additional Table and Figures

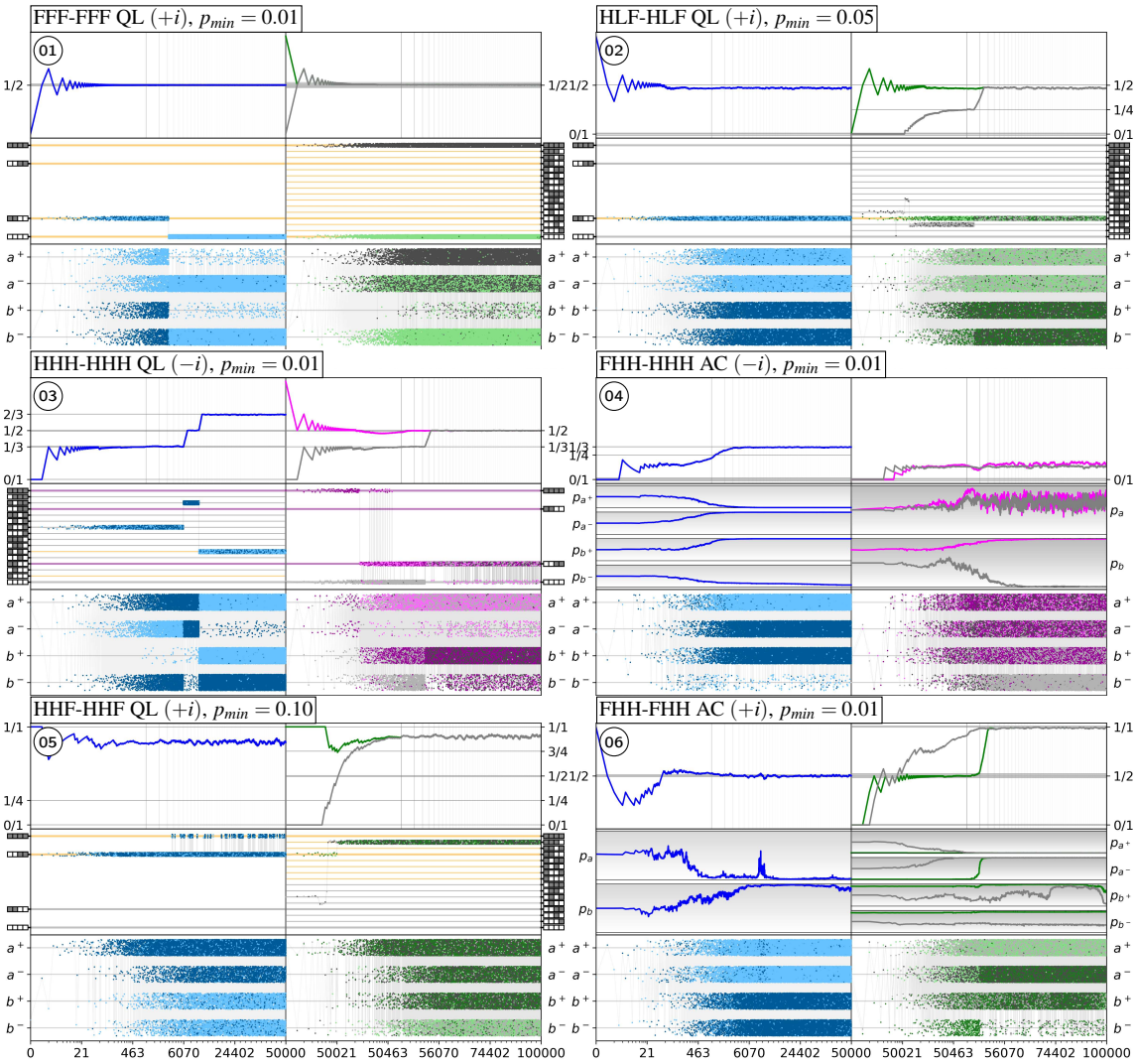


Figure A.1: Example runs presented in Figure 7.4 with additional information. On every plot, the top panel is identical to the plot in Figure 7.4. The bottom panels represent the corresponding agent trajectory within the environment states space. Light dots mean the agent has chosen action ○ and dark dots mean action ●. Middle panels represent the search trajectory within the policies space $\mathscr{P}$ and $\bar{\mathscr{P}}$. When searching $\mathscr{P}_{det}$ and $\bar{\mathscr{P}}_{det}$ with the QL method, each horizontal line represents one deterministic policy labelled with corresponding actions in canonical order. Optimal policies within $\mathscr{P}^*_{det}$ are colored gold, and diminished optimal policies within $\bar{\mathscr{P}}^*_{det}$ are colored purple. When searching $\mathscr{P}_{sto}$ and $\bar{\mathscr{P}}_{sto}$ with the AC method, each trajectory corresponds to one dimension of the stochastic search space, *i.e.* the probability of choosing action ● depending on current observation. Figure continued in Figure A.2

joint profile	meas.	res. stde	joint profile	meas.	res. stde
FFF-FFF	LT	0.0000017	HLF-HLF	LT	0.0576113
FFF-FFF	IT	0.0001975	HLF-HLF	IT	0.0369440
HFF-HHF	LT	0.0075877	FHH-FHH	LT	0.0260944
HFF-HHF	IT	0.0235293	FHH-FHH	IT	0.1203736
FHF-HHF	LT	0.0094990	FHH-HHH	LT	0.0383201
FHF-HHF	IT	0.0263794	FHH-HHH	IT	0.1690942
HHF-HHF	LT	0.0405426	HHH-HHH	LT	0.0443568
HHF-HHF	IT	0.0617372	HHH-HHH	IT	0.1214023

Table A.2: Residual standard error of the linear models fitted for measures LT and IT under the various joint profiles.

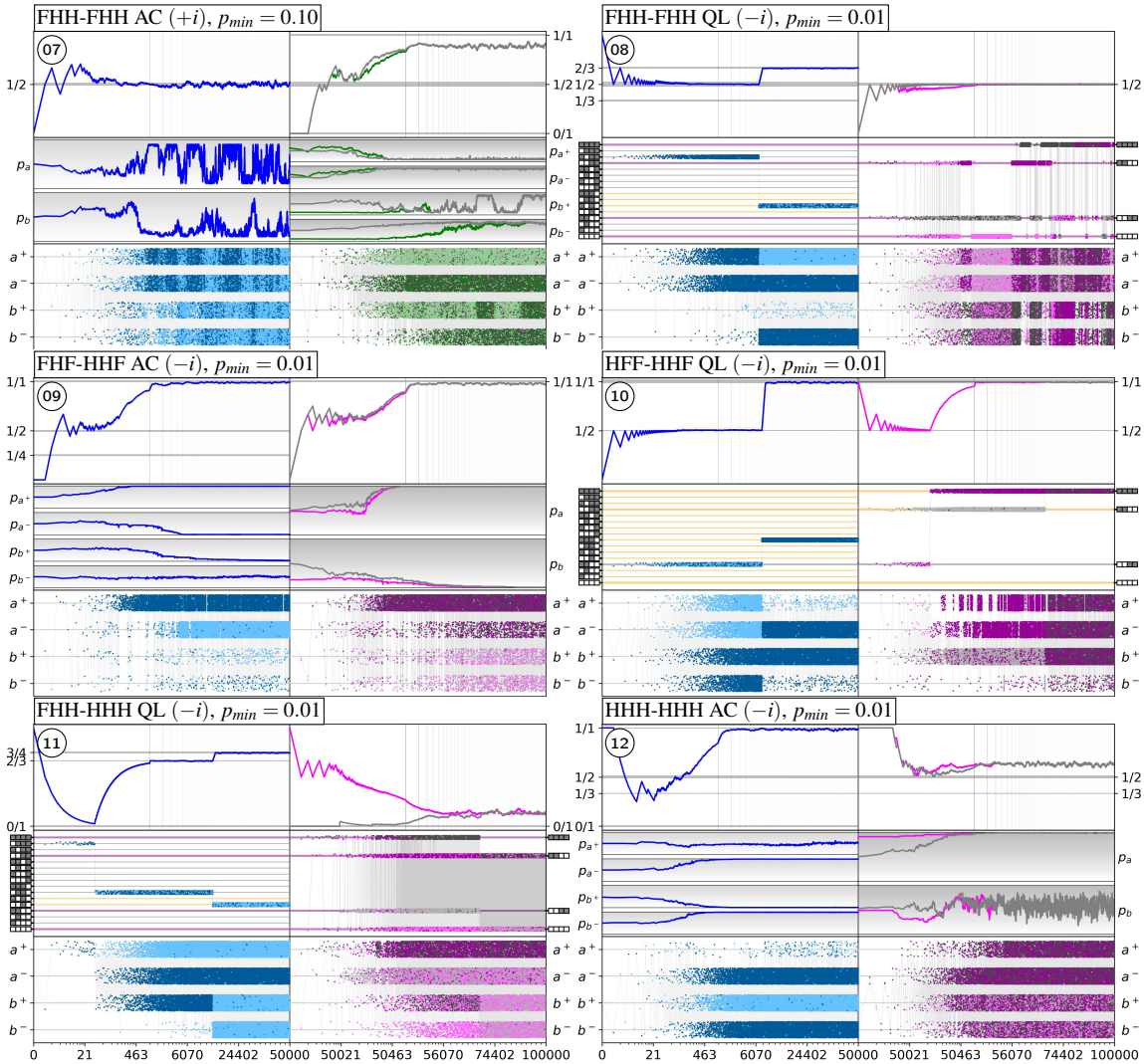


Figure A.2: Figure A.1 continued.

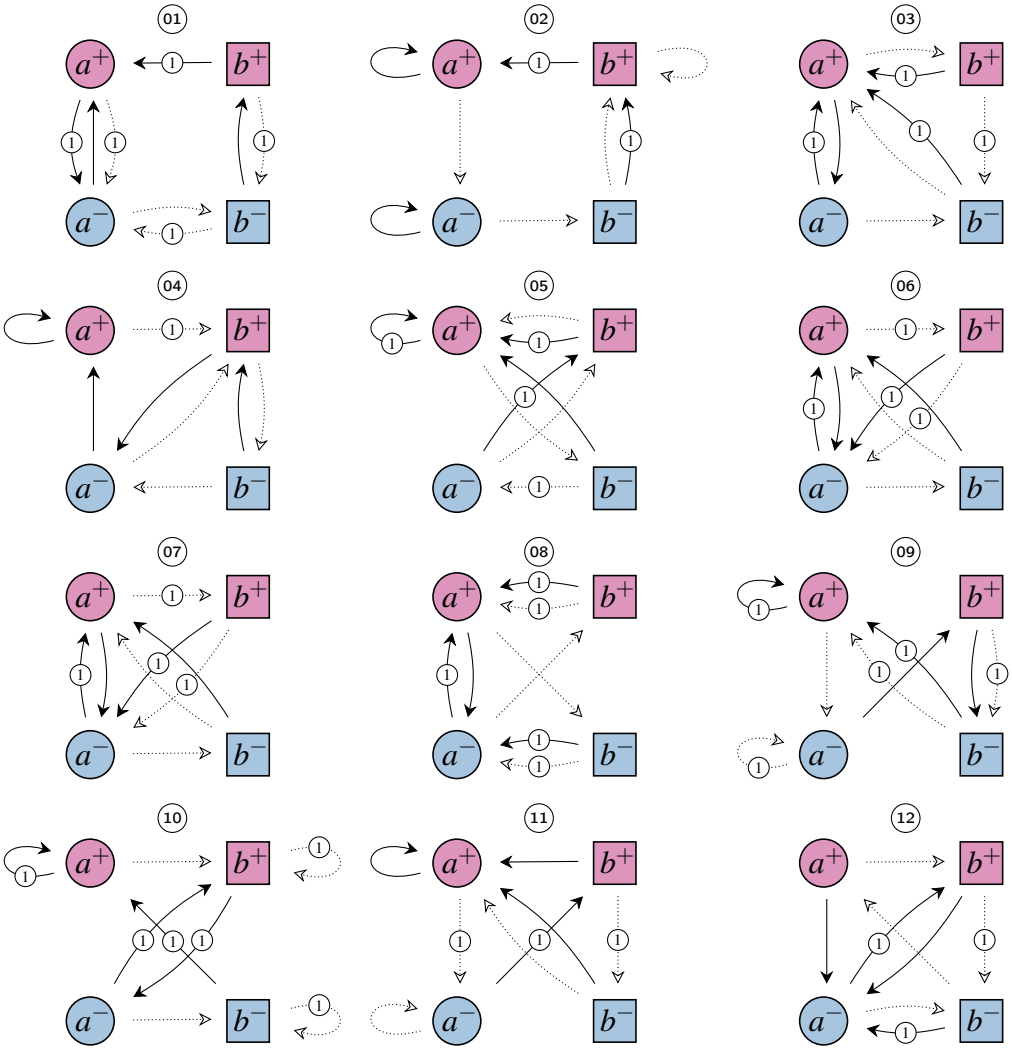


Figure A.3: Environments used in the example runs represented in Figure 7.4.

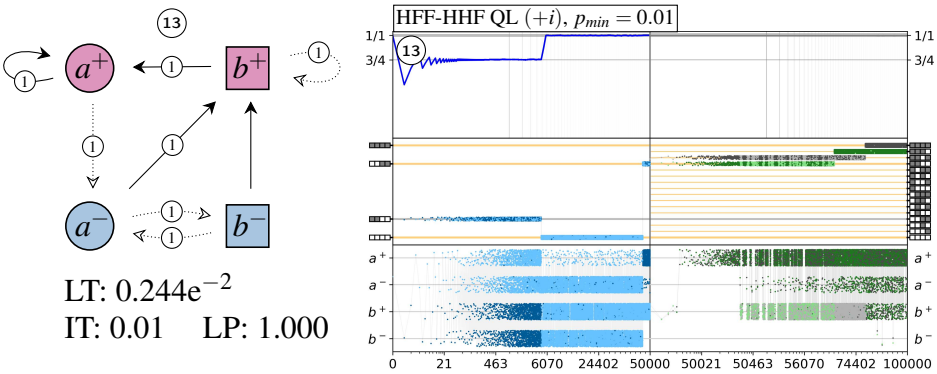


Figure A.4: Example run illustrating effect **(G)**.

Bibliography

[1] Ababsa, Tarek, Djedi, Noureddine, Duthen, Yves, and Cussat-Blanc, Sylvain. *Splittable Metamorphic Carrier Robots*. In: 14th International Conference on the Synthesis and Simulation of Living Systems, ALIFE. New York, US: The MIT Press, 2014, pp. 1–8.

[2] Abramowitz, Milton and Stegun, Irene A. *Lengendre Functions*. In: Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables. 9th. New York: Dover, 1972, pp. 771–802.

[3] Bacon, Pierre-Luc, Harb, Jean, and Precup, Doina. *The Option-Critic Architecture*. In: AAAI Conference on Artificial Intelligence 31.1 (2017).

[4] Bakker, Bram. *Reinforcement Learning with Long Short-Term Memory*. In: 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 1475–1482.

[5] Bakker, Bram, Zhumatiy, V., Gruener, G., and Schmidhuber, J. *Quasi-Online Reinforcement Learning for Robots*. In: IEEE International Conference on Robotics and Automation. ICRA 2006. Orlando, FL, USA: IEEE, 2006, pp. 2997–3002.

[6] Bandura, Albert and Walters, R.H. *Social Learning and Personality Development.* Social Learning and Personality Development. Holt Rinehart and Winston: New York, 1963.

[7] Barreto, André, Borsa, Diana, Quan, John, Schaul, Tom, Silver, David, Hessel, Matteo, Mankowitz, Daniel, Zidek, Augustin, and Munos, Remi. *Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement*. In: 35th International Conference on Machine Learning. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 501–510.

[8] Barreto, André, Munos, Rémi, Schaul, Tom, and Silver, David. *Successor Features for Transfer in Reinforcement Learning*. In: Advances in Neural Information Processing Systems. Vol. 30. Curran Associates, Inc., 2017, pp. 4055–4065.

[9] Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. *Curriculum Learning*. In: 26th Annual International Conference on Machine Learning - ICML '09. Montreal, Quebec, Canada: ACM Press, 2009, pp. 1–8.

[10] Berner, Christopher *et al. Dota 2 with Large Scale Deep Reinforcement Learning*. In: CoRR abs/1912.06680 (2019).

[11] Bonnici, Iago, Gouaïch, Abdelkader, and Michel, Fabien. *Effects of Input Addition in Learning for Adaptive Games: Towards Learning with Structural Changes*. In: EvoApplications: Applications of Evolutionary Computation. Vol. LNCS. Leipzig, Germany, 2019, pp. 172–184.

[12] Bonnici, Iago, Gouaïch, Abdelkader, and Michel, Fabien. *Input Addition and Deletion in Reinforcement: Towards Learning with Structural Changes*. In: 19th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '20. Auckland, New Zealand: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 177–185.

[13] Boots, Byron and Gordon, Geoffrey J. *Predictive State Temporal Difference Learning*. In: Advances in neural information processing systems. Curran Associates, Inc. 23 (2010). In collab. with J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, pp. 271–279.

[14] Busto, Pau Panareda and Gall, Juergen. *Open Set Domain Adaptation*. In: International Conference on Computer Vision (ICCV). Venice: IEEE, 2017, pp. 754–763.

[15] Caruana, Rich. *Learning Many Related Tasks at the Same Time with Backpropagation*. In: 7th International Conference on Neural Information Processing Systems. NIPS'94. Denver, Colorado: MIT Press, 1994, pp. 657–664.

[16] Pu-Cheng, Zhou, Bing-Rong, Hong, Qing-Cheng, Huang, and Khurshid, Javaid. *Hybrid Multiagent Reinforcement Learning Approach: The Pursuit Problem*. In: Information Technology Journal 5.6 (2006), pp. 1006–1011.

[17] Chiplunkar, Ankit, Rachelson, Emmanuel, Colombo, Michele, and Morlier, Joseph. *Approximate Inference in Related Multi-Output Gaussian Process Regression*. In: Pattern Recognition Applications and Methods. Ed. by Ana Fred, Maria De Marsico, and Gabriella Sanniti di Baja. Vol. 10163. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 88–103.

[18] Cho, Kyunghyun, van Merrienboer, Bart, Gulcehre, Caglar, Bougares, Fethi, Schwenk, Holger, Bengio, Yoshua, and Bahdanau, Dzmitry. *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*. In: Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724–1734.

[19] Clavera, Ignasi, Held, David, and Abbeel, Pieter. *Policy Transfer via Modularity and Reward Guiding*. In: International Conference on Intelligent Robots and Systems (IROS). 2017.

[20] Cornuéjols, Antoine, Koriche, Frédéric, and Nock, Richard. *Statistical Computational Learning*. In: A Guided Tour of Artificial Intelligence Research. Ed. by Pierre Marquis, Odile Papini, and Henri Prade. Cham: Springer International Publishing, 2020, pp. 341–388.

[21] Cornuéjols, Antoine, Murena, Pierre-Alexandre, and Olivier, Raphaël. *Transfer Learning by Learning Projections from Target to Source*. In: Advances in Intelligent Data Analysis XVIII. Ed. by Michael R. Berthold, Ad Feelders, and Georg Krempl. Vol. 12080. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 119–131.

[22] Cui, Yuwei, Ahmad, Subutai, and Hawkins, Jeff. *Continuous Online Sequence Learning with an Unsupervised Neural Network Model*. In: Neural Computation 28.11 (2016), pp. 2474–2504.

[23] Cutler, Mark and How, Jonathan P. *Efficient Reinforcement Learning for Robots Using Informative Simulated Priors*. In: International Conference on Robotics and Automation (ICRA). Seattle, WA, USA: IEEE, 2015, pp. 2605–2612.

[24] Darwin, Charles. *On the Origins of Species by Means of Natural Selection*. In: London: Murray (1859), p. 247.

[25] De Rosario-Martinez, Helios. *Phia: Post-Hoc Interaction Analysis*. 2015.

[26] Dempster, Arthur P., Laird, Nan M., and Rubin, Donald B. *Maximum Likelihood from Incomplete Data via the EM Algorithm*. In: Journal of the Royal Statistical Society. Series B (Methodological) 39.1 (1977), pp. 1–38.

[27] Devin, Coline, Gupta, Abhishek, Darrell, Trevor, Abbeel, Pieter, and Levine, Sergey. *Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer*. In: International Conference on Robotics and Automation (ICRA). Singapore, Singapore: IEEE, 2017, pp. 2169–2176.

[28] Dhillon, Anamika and Verma, Gyanendra K. *Convolutional Neural Network: A Review of Models, Methodologies and Applications to Object Detection*. In: Progress in Artificial Intelligence 9.2 (2020), pp. 85–112.

[29] Doncieux, Stephane, Bredeche, Nicolas, Mouret, Jean-Baptiste, and Eiben, Agoston E. (Gusz). *Evolutionary Robotics: What, Why, and Where To*. In: Frontiers in Robotics and AI 2 (2015).

[30] Elgammal, Ahmed M., Mazzone, Marian, Liu, Bingchen, Kim, Diana, and Elhoseiny, Mohamed. *The Shape of Art History in the Eyes of the Machine*. In: AAAI Conference on Artificial Intelligence 32.1 (2018).

[31] Elman, Jeffrey L. *Finding Structure in Time*. In: Cognitive Science 14.2 (1990), pp. 179–211.

[32] Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. In: 34th International Conference on Machine Learning (Sydney, NSW, Australia). Vol. 70. JMLR.org, 2017, pp. 1126–1135.

[33] Francillette, Yannick. *Modèle Adaptatif d'activités Pour Les Jeux Ubiquitaires*. PhD Thesis. LIRMM: Université de Montpellier, 2014.

[34] Frans, Kevin, Ho, Jonathan, Chen, Xi, Abbeel, Pieter, and Schulman, John. *Meta Learning Shared Hierarchies*. In: ArXiv abs/1710.09767 (2018).

[35] Friedman, Jerome H. *Multivariate Adaptive Regression Splines*. In: The Annals of Statistics 19.1 (1991), pp. 1–67.

[36] Gama, João, Žliobaitė, Indrė, Bifet, Albert, Pechenizkiy, Mykola, and Bouchachia, Abdelhamid. *A Survey on Concept Drift Adaptation*. In: ACM Computing Surveys 46.4 (2014), pp. 1–37.

[37] Garcia, Frédérick and Rachelson, Emmanuel. *Markov Decision Processes*. In: Markov Decision Processes in Artificial Intelligence. Ed. by Olivier Sigaud and Olivier Buffet. Hoboken, NJ USA: John Wiley & Sons, Inc., 2013, pp. 1–38.

[38] Ge, Liang, Gao, Jing, Ngo, Hung, Li, Kang, and Zhang, Aidong. *On Handling Negative Transfer and Imbalanced Distributions in Multiple Source Transfer Learning: Multiple Source Transfer Learning*. In: Statistical Analysis and Data Mining: The ASA Data Science Journal 7.4 (2014), pp. 254–271.

[39] Al-Ghossein, Marie, Murena, Pierre-Alexandre, Cornuéjols, Antoine, and Abdessalem, Talel. *Online Learning with Reoccurring Drifts: The Perspective of Case-Based Reasoning*. In: 3rd Workshop on Synergies between CBR and Data Mining, ICCBR. Stockholm, Sweden, 2018.

[40] Goddeau, David and Pineau, Joelle. *Fast Reinforcement Learning of Dialog Strategies*. In: International Conference on Acoustics, Speech, and Signal Processing. Vol. 2. Istanbul, Turkey: IEEE, 2000, pp. II1233–II1236.

[41] Goldberg, Yoav. *A Primer on Neural Network Models for Natural Language Processing*. In: Journal of Artificial Intelligence Research 57 (2016), pp. 345–420.

[42] Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya, and Levine, Sergey. *Continuous Deep Q-Learning with Model-Based Acceleration*. In: 33rd International Conference on Machine Learning. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 2829–2838.

[43] Gupta, Anil K., Smith, Ken G., and Shalley, Christina E. *The Interplay Between Exploration and Exploitation*. In: Academy of Management Journal 49.4 (2006), pp. 693–706.

[44] Hanna, Christopher J., Hickey, Raymond J., Charles, Darryl K., and Black, Michaela M. *Modular Reinforcement Learning Architectures for Artificially Intelligent Agents in Complex Game Environments*. In: Symposium on Computational Intelligence and Games (CIG). Copenhagen, Denmark: IEEE, 2010, pp. 380–387.

[45] Harel, Maayan and Mannor, Shie. *Learning from Multiple Outlooks*. In: 28th International Conference on International Conference on Machine Learning. ICML'11. Madison, WI, USA: Omnipress, 2011, pp. 401–408.

[46] Haykin, Simon. *Neural Networks: A Comprehensive Foundation*. Second. OCLC: 643435359. Delhi: Pearson Education, 1999.

[47] Heng Wang and Abraham, Zubin. *Concept Drift Detection for Streaming Data*. In: 2015 International Joint Conference on Neural Networks (IJCNN). Killarney, Ireland: IEEE, 2015, pp. 1–9.

[48] Hester, Todd and Stone, Peter. *TEXPLORE: Real-Time Sample-Efficient Reinforcement Learning for Robots*. In: Machine Learning 90.3 (2013), pp. 385–429.

[49] Hlynsson, Hlynur, Escalante-B., Alberto, and Wiskott, Laurenz. *Measuring the Data Efficiency of Deep Learning Methods:* in: 8th International Conference on Pattern Recognition Applications and Methods. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2019, pp. 691–698.

[50] Hochreiter, Sepp and Schmidhuber, Jürgen. *Long Short-Term Memory*. In: Neural Computation 9.8 (1997), pp. 1735–1780.

[51] Hsu, Feng-hsiung. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton: Princeton University Press, 2002. 298 pp.

[52] Jaber, Ghazal, Cornuéjols, Antoine, and Tarroux, Philippe. *Online Learning: Searching for the Best Forgetting Strategy under Concept Drift*. In: Neural Information Processing. Ed. by Minho Lee, Akira Hirose, Zeng-Guang Hou, and Rhee Man Kil. Red. by David Hutchison *et al.* Vol. 8227. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 400–408.

[53] Jain, Lakhmi C., Seera, Manjeevan, Lim, Chee Peng, and Balasubramaniam, P. *A Review of Online Learning in Supervised Neural Networks*. In: Neural Computing and Applications 25.3-4 (2014), pp. 491–509.

[54] Jin, Yu, Duffield, Nick, Erman, Jeffrey, Haffner, Patrick, Sen, Subhabrata, and Zhang, Zhi-Li. *A Modular Machine Learning System for Flow-Level Traffic Classification in Large Networks*. In: ACM Transactions on Knowledge Discovery from Data 6.1 (2012), pp. 1–34.

[55] Jolliffe, Ian. *Principal Component Analysis*. In: International Encyclopedia of Statistical Science. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096.

[56] Jones, M. Chris, Marron, James S., and Sheather, Simon J. *A Brief Survey of Bandwidth Selection for Density Estimation*. In: Journal of the American Statistical Association 91.433 (1996), pp. 401–407.

[57] Jung, Ralf, Jourdan, Jacques-Henri, Krebbers, Robbert, and Dreyer, Derek. *Safe Systems Programming in Rust*. In: Communications of the ACM 64.4 (2021), pp. 144–152.

[58] Kalmár, Zsolt, Szepesvári, Csaba, and Lorincz, András. *Modular Reinforcement Learning: An Application to a Real Robot Task*. In: Learning Robots. Ed. by Andreas Birk and John Demiris. Red. by G. Goos, J. Hartmanis, and J. van Leeuwen. Vol. 1545. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 29–45.

[59] Kaplanis, Christos, Shanahan, Murray, and Clopath, Claudia. *Continual Reinforcement Learning with Complex Synapses*. In: 35th International Conference on Machine Learning. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 2497–2506.

[60] Karras, Tero, Aila, Timo, Laine, Samuli, and Lehtinen, Jaakko. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. In: 6th International Conference on Learning Representations (ICLR). Ed. by Yann LeCun and Yoshua Bengio. Vancouver, British Columbia, Canada, 2018.

[61] Khanıev, Tahır A., Unver, İhsan, and Maden, Selahattın. *On the Semi-Markovian Random Walk with Two Reflecting Barriers*. In: Stochastic Analysis and Applications 19.5 (2001), pp. 799–819.

[62] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*. In: 3rd International Conference on Learning Representations (ICLR). Ed. by Yoshua Bengio and Yann LeCun. San Diego, CA, USA, 2015.

[63] Kirkpatrick, James *et al. Overcoming Catastrophic Forgetting in Neural Networks*. In: Proceedings of the National Academy of Sciences 114.13 (2017), pp. 3521–3526.

[64] Kirkpatrick, Scott, Gelatt, C. Daniel, and Vecchi, Mario P. *Optimization by Simulated Annealing*. In: Science 220.4598 (1983), pp. 671–680.

[65] Kohonen, Teuvo. *Self-Organizing Maps*. 3rd ed. Springer Series in Information Sciences 30. Berlin ; New York: Springer, 2001. 501 pp.

[66] Kotsiantis, Sotiris B. *Supervised Machine Learning: A Review of Classification Techniques*. In: Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies. Amsterdam, Netherlands: IOS Press, 2007, pp. 3–24.

[67] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. *ImageNet Classification with Deep Convolutional Neural Networks*. In: Advances in Neural Information Processing Systems 25. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[68] Lazaric, Alessandro. *Transfer in Reinforcement Learning: A Framework and a Survey*. In: Reinforcement Learning. Ed. by Marco Wiering and Martijn van Otterlo. Vol. 12. Springer, 2012, pp. 143–173.

[69] Lecarpentier, Erwan and Rachelson, Emmanuel. *Non-Stationary Markov Decision Processes, a Worst-Case Approach Using Model-Based Reinforcement Learning*. In: Advances in Neural Information Processing Systems. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019, pp. 7216–7225.

[70] Lecun, Yann and Bengio, Yoshua. *Convolutional networks for images, speech, and time-series*. In: The handbook of brain theory and neural networks. Ed. by M.A. Arbib. MIT Press, 1995.

[71] LeCun, Yann, Denker, John S., and Solla, Sara A. *Optimal Brain Damage*. In: Advances in Neural Information Processing Systems 2. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 598–605.

[72] Lee, Juneyoung, Kim, Yoonseung, Song, Youngju, Hur, Chung-Kil, Das, Sanjoy, Majnemer, David, Regehr, John, and Lopes, Nuno P. *Taming Undefined Behavior in LLVM*. In: 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). Barcelona, Spain: ACM Press, 2017, pp. 633–647.

[73] Lee, Kai-Fu and Mahajan, Sanjoy. *Corrective and Reinforcement Learning for Speaker-Independent Continuous Speech Recognition*. In: Computer Speech & Language 4.3 (1990), pp. 231–245.

[74] Levine, Sergey and Koltun, Vladlen. *Guided Policy Search*. In: ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Atlanta, Georgia, USA: PMLR, 2013, pp. 1–9.

[75] Levine, Sergey, Kumar, Aviral, Tucker, George, and Fu, Justin. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. In: (2020).

[76] Li, Wen, Duan, Lixin, Xu, Dong, and Tsang, Ivor W. *Learning with Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation*. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 36.6 (2014), pp. 1134–1148.

[77] Liao, X.X. and Mao, Xuerong. *Exponential Stability and Instability of Stochastic Neural Networks*. In: Stochastic Analysis and Applications 14.2 (1996), pp. 165–185.

[78] Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexand er, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. *Continuous Control with Deep Reinforcement Learning*. In: arXiv e-prints (2015), arXiv:1509.02971.

[79] Lin, Long-Ji and Mitchell, Tom M. *Reinforcement Learning with Hidden States*. In: 2nd International Conference on From Animals to Animats: Simulation of Adaptive Behavior: Simulation of Adaptive Behavior. Honolulu, Hawai, USA: MIT Press, 1993, pp. 271–280.

[80] Lipton, Zachary Chase. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. In: CoRR abs/1506.00019 (2015).

[81] Lloyd, Stuart. *Least Squares Quantization in PCM*. In: IEEE Transactions on Information Theory 28.2 (1982), pp. 129–137.

[82] Lopez-Paz, David and Ranzato, Marc'Aurelio. *Gradient Episodic Memory for Continuum Learning*. In: Advances in Neural Information Processing Systems. Vol. 30. Curran Associates, Inc., 2017, pp. 6467–6476.

[83] Losing, Viktor, Hammer, Barbara, and Wersing, Heiko. *Incremental On-Line Learning: A Review and Comparison of State of the Art Algorithms*. In: Neurocomputing 275 (2018), pp. 1261–1274.

[84] Mandic, Danilo P. and Chambers, Jonathon A. *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures, and Stability*. Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications, and Control. Chichester ; New York: John Wiley, 2001. 285 pp.

[85] Maniezzo, Alberto Colorni Marco Dorigo Vittorio. *Distributed Optimization by Ant Colonies*. In: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. Mit Press, 1992, p. 134.

[86] McCulloch, Warren S. and Pitts, Walter. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. In: The Bulletin of Mathematical Biophysics 5.4 (1943), pp. 115–133.

[87] Mhaskar, Hrushikesh N. and Poggio, Tomaso. *Deep vs. Shallow Networks: An Approximation Theory Perspective*. In: Analysis and Applications 14.06 (2016), pp. 829–848.

[88] Mitchell, Tom M. *Machine Learning*. Vol. 45. McGraw-Hill Series in Computer Science. New York: McGraw-Hill, 1997. 414 pp.

[89] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin A. *Playing Atari with Deep Reinforcement Learning*. In: CoRR abs/1312.5602 (2013).

[90] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., *et al. Human-Level Control through Deep Reinforcement Learning*. In: Nature 518.7540 (2015), pp. 529–533.

[91] Moriarty, David E., Schultz, Allan C., and Grefenstette, John J. *Evolutionary Algorithms for Reinforcement Learning*. In: Journal of Artificial Intelligence Research 11 (1999), pp. 241–276.

[92] Narvekar, Sanmit, Peng, Bei, Leonetti, Matteo, Sinapov, Jivko, Taylor, Matthew E., and Stone, Peter. *Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey*. In: Journal of Machine Learning Research 21.181 (2020), pp. 1–50.

[93] Nichols, Carol and Safari, an O'Reilly Media Company. *The Rust Programming Language (Covers Rust 2018)*. OCLC: 1119061124. 2019.

[94] Ono, Norihiko and Fukumoto, Kenji. *Multi-Agent Reinforcement Learning: A Modular Approach*. In: 2nd International Conference on Multiagent Systems, AAAI (1996), p. 7.

[95] Osman, Ibrahim H. and Laporte, Gilbert. *Metaheuristics: A Bibliography*. In: Annals of Operations Research 63.5 (1996), pp. 511–623.

[96] Partalas, Ioannis, Feneris, Ioannis, and Vlahavas, Ioannis. *A Hybrid Multiagent Reinforcement Learning Approach Using Strategies and Fusion*. In: International Journal on Artificial Intelligence Tools 17.05 (2008), pp. 945–962.

[97] Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. *On the Difficulty of Training Recurrent Neural Networks*. In: 30th International Conference on International Conference on Machine Learning (Atlanta, GA, USA). Vol. 28. ICML'13. JMLR.org, 2013, pp. III-1310-III–1318.

[98] Paszke, Adam, Gross, Sam, Chintala, Soumith, Chanan, Gregory, Yang, Edward, DeVito, Zachary, Lin, Zeming, Desmaison, Alban, Antiga, Luca, and Lerer, Adam. *Automatic Differentiation in PyTorch*. 2017.

[99] Potter, Mitchell A. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD Thesis. Fairfax, VA, USA: George Mason University, 1997.

[100] Prentice, Ross L. and Pyke, Ronald. *Logistic Disease Incidence Models and Case-Control Studies*. In: Biometrika 66.3 (1979), pp. 403–411.

[101] R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2020.

[102] Rachelson, Emmanuel, Garcia, Frédérick, and Fabiani, Patrick. *Extending the Bellman Equation for MDPs to Continuous Actions and Continuous Time in the Discounted Case*. In: International Symposium on Artificial Intelligence and Mathematics. Fort Lauderdale, USA, 2008.

[103] Reinsel, David, Gantz, John, and Rydning, John. *The Digitization of the World, From Edge to Core*. IDC, 2018.

[104] Ring, Mark B. *Continual Learning in Reinforcement Environments*. PhD Thesis. Austin, TX, USA: University of Texas at Austin, 1994.

[105] Ring, Mark B. *CHILD: A First Step Towards Continual Learning*. In: Machine Learning 28.1 (1997), pp. 77–104.

[106] Rohlfshagen, Philipp, Lehre, Per Kristian, and Yao, Xin. *Dynamic Evolutionary Optimisation: An Analysis of Frequency and Magnitude of Change*. In: 11th Annual Conference on Genetic and Evolutionary Computation - GECCO '09. Montreal, Québec, Canada: ACM Press, 2009, p. 1713.

[107] Rusu, Andrei A., Rabinowitz, Neil C., Desjardins, Guillaume, Soyer, Hubert, Kirkpatrick, James, Kavukcuoglu, Koray, Pascanu, Razvan, and Hadsell, Raia. *Progressive Neural Networks*. In: CoRR abs/1606.04671 (2016).

[108] Sastry, Kumara, Goldberg, David E., and Kendall, Graham. *Genetic Algorithms*. In: Search Methodologies. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2014, pp. 93–117.

[109] Schmidhuber, Jürgen. *Deep Learning in Neural Networks: An Overview*. In: Neural Networks 61 (2015), pp. 85–117.

[110] Schwartz, Anton. *A Reinforcement Learning Method for Maximizing Undiscounted Rewards*. In: ICML. 1993, pp. 298–305.

[111] Scott, David W. *Kernel Density Estimation*. In: Wiley StatsRef: Statistics Reference Online. Ed. by N. Balakrishnan, Theodore Colton, Brian Everitt, Walter Piegorsch, Fabrizio Ruggeri, and Jozef L. Teugels. Chichester, UK: John Wiley & Sons, Ltd, 2018, pp. 1–7.

[112] Seff, Ari, Beatson, Alex, Suo, Daniel, and Liu, Han. *Continual Learning in Generative Adversarial Nets*. In: CoRR abs/1705.08395 (2017).

[113] Sigaud, Olivier and Droniou, Alain. *Towards Deep Developmental Learning*. In: IEEE Transactions on Cognitive and Developmental Systems 8.2 (2016), pp. 99–114.

[114] Sigaud, Olivier and Stulp, Freek. *Policy Search in Continuous Action Domains: An Overview*. In: Neural Networks 113 (2019), pp. 28–40.

[115] Silver, David *et al. Mastering the Game of Go with Deep Neural Networks and Tree Search*. In: Nature 529 (2016), pp. 484–503.

[116] Smart, Williiam D. and Pack Kaelbling, Leslie. *Effective Reinforcement Learning for Mobile Robots*. In: International Conference on Robotics and Automation. Vol. 4. Washington, DC, USA: IEEE, 2002, pp. 3404–3410.

[117] Sodhani, Shagun, Chandar, Sarath, and Bengio, Yoshua. *Toward Training Recurrent Neural Networks for Lifelong Learning*. In: Neural Computation 32.1 (2020), pp. 1–35.

[118] Spaan, Matthijs T. J. *Partially Observable Markov Decision Processes*. In: Reinforcement Learning: State-of-the-Art. Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414.

[119] Spooner, Thomas, Fearnley, John, Savani, Rahul, and Koukorinis, Andreas. *Market Making via Reinforcement Learning*. In: 17th International Conference on Autonomous Agents and MultiAgent Systems. AAMAS '18. Stockholm, Sweden: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 434–442.

[120] Sprague, Nathan and Ballard, Dana. *Multiple-Goal Reinforcement Learning with Modular Sarsa(O)*. In: 18th International Joint Conference on Artificial Intelligence. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1445–1447.

[121] Stanley, Kenneth O. and Miikkulainen, Risto. *Efficient Reinforcement Learning Through Evolving Neural Network Topologies*. In: 4th Annual Conference on Genetic and Evolutionary Computation. GECCO'02. New York City, New York: Morgan Kaufmann Publishers Inc., 2002, pp. 569–577.

[122] Suro, François. *Epigenetic Learning of Autonomous Behaviours in a Society of Agents*. PhD Thesis. Université de Montpellier, 2020. 193 pp.

[123] Suro, François, Ferber, Jacques, Stratulat, Tiberiu, and Michel, Fabien. *A Hierarchical Representation of Behaviour Supporting Open Ended Development and Progressive Learning for Artificial Agents*. In: Autonomous Robots (2021).

[124] Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. 2nd. Adaptive Computation and Machine Learning Series. Cambridge, Mass: MIT Press, 2018. 322 pp.

[125] Sutton, Richard S., Modayil, Joseph, Delp, Michael, Degris, Thomas, Pilarski, Patrick M., White, Adam, and Precup, Doina. *Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction*. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. AAMAS '11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 761–768.

[126] Tallec, Corentin and Ollivier, Yann. *Unbiased Online Recurrent Optimization*. In: International Conference on Learning Representations. 2018.

[127] Tanaka, Fumihide and Yamamura, Masayuki. *Multitask Reinforcement Learning on the Distribution of MDPs*. In: International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium. Vol. 3. Kobe, Japan: IEEE, 2003, pp. 1108–1113.

[128] Taylor, Matthew E. and Stone, Peter. *Transfer Learning for Reinforcement Learning Domains: A Survey.* In: Journal of Machine Learning Research 10.7 (2009), pp. 1633–1685.

[129] Teh, Yee, Bapst, Victor, Czarnecki, Wojciech M., Quan, John, Kirkpatrick, James, Hadsell, Raia, Heess, Nicolas, and Pascanu, Razvan. *Distral: Robust Multitask Reinforcement Learning*. In: Advances in Neural Information Processing Systems 30. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4496–4506.

[130] Thrun, Sebastian. *Is Learning the N-Th Thing Any Easier Than Learning the First?* In: 8th International Conference on Neural Information Processing Systems. NIPS'95. Denver, Colorado: MIT Press, 1995, pp. 640–646.

[131] Torrey, Lisa and Shavlik, Jude. *Transfer Learning*. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques. Ed. by Emilio Soria Olivas, José David Martín Guerrero, Marcelino Martinez-Sober, Jose Rafael Magdalena-Benedito, and Antonio José Serrano López. IGI Global, 2010, pp. 242–264.

[132] Tsymbal, Alexey. *The Problem of Concept Drift: Definitions and Related Work*. In: (2004).

[133] Uchibe, Eiji, Asada, Minoru, and Hosoda, Koh. *Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning*. In: International Conference on Intelligent Robots and Systems. IROS '96. Vol. 3. Osaka, Japan: IEEE/RSJ, 1996, pp. 1329–1336.

[134] Van Rossum, Guido and Drake, Fred L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[135] Watanabe, Chihiro, Hiramatsu, Kaoru, and Kashino, Kunio. *Modular Representation of Layered Neural Networks*. In: Neural Networks 97 (2018), pp. 62–73.

[136] Watkins, Christopher J. C. H. *Learning from Delayed Rewards*. In: PhD thesis, Cambridge University (1989).

[137] Webb, Geoffrey I., Lee, Loong Kuan, Petitjean, François, and Goethals, Bart. *Understanding Concept Drift*. In: CoRR abs/1704.00362 (2017).

[138] Widmer, Gerhard and Kubat, Miroslav. *Learning in the Presence of Concept Drift and Hidden Contexts*. In: Machine Learning 23.1 (1996), pp. 69–101.

[139] Wladawsky-Berger, Irving. *Conceptualizing AI in Human Terms Is Misleading*. In: The Wall Street Journal (2020).

[140] Wolfram Research, Inc. *Mathematica, Version 11.3*. 2018.

[141] Xu, Ju and Zhu, Zhanxing. *Reinforced Continual Learning*. In: Advances in Neural Information Processing Systems 31. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018, pp. 899–908.

[142] Yang, Qiang, Chen, Yuqiang, Xue, Gui-Rong, Dai, Wenyuan, and Yu, Yong. *Heterogeneous Transfer Learning for Image Clustering via the Social Web*. In: Joint Conference of the 47th Annual Meeting of the ACL and 4th International Joint Conference on Natural Language Processing of the AFNLP. Vol. 1. ACL '09. Suntec, Singapore: Association for Computational Linguistics, 2009, pp. 1–9.

[143] Yaochu Jin and Sendhoff, B. *Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies*. In: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38.3 (2008), pp. 397–415.

[144] Zenke, Friedemann, Poole, Ben, and Ganguli, Surya. *Continual Learning Through Synaptic Intelligence*. In: 34th International Conference on Machine Learning. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 3987–3995.

[145] Zentall, Thomas R. *Perspectives on Observational Learning in Animals.* In: Journal of Comparative Psychology 126.2 (2012), pp. 114–128.

[146] Zhang, Yu, Chan, William, and Jaitly, Navdeep. *Very Deep Convolutional Networks for End-to-End Speech Recognition*. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP). New Orleans, LA: IEEE, 2017, pp. 4845–4849.

[147] Zhou, Ding-Xuan. *Universality of Deep Convolutional Neural Networks*. In: Applied and Computational Harmonic Analysis 48.2 (2020), pp. 787–794.

[148] Žliobaitė, Indrė, Pechenizkiy, Mykola, and Gama, João. *An Overview of Concept Drift Applications*. In: Big Data Analysis: New Algorithms for a New Society. Ed. by Nathalie Japkowicz and Jerzy Stefanowski. Vol. 16. Cham: Springer International Publishing, 2016, pp. 91–114.

[149] Zoph, Barret and Le, Quoc V. *Neural Architecture Search with Reinforcement Learning*. In: CoRR abs/1611.01578 (2016).

List of Figures

1.1 The Imaginary Sticky Roverbot 19

2.1 Various categories of behavioural searches. 36
2.2 Overview of ML as an Euler Diagram 44

4.1 Illustration of Various TL Contexts 69

5.1 Example Multiple Streams 84
5.2 IPL Profiles 94
5.3 Global design of the Experiments 99

6.1 Example Synthetic Streams 103
6.2 Example Learning Curves in OSL 110
6.3 Violin Plots for Measures in OSL: input addition 111
6.4 Violin Plots for Measures in OSL: input deletion 112

7.1 Example transition functions from the benchmark and their diminished projections. 124
7.2 Diminished Stochastic and Deterministic Search Spaces 138
7.3 Violin Plots for Measures in RL 149
7.4 Example Runs 150

A.1 Extended Visualization of Example Runs 177
A.2 Figure A.1 continued. 178
A.3 Environments Used in Example Runs 179
A.4 Example run illustrating effect **(G)**. 179

List of Tables

4.1 Overview of Related Works 66

7.1 Joint profiles partitioning the $\mathscr{T}_{\text{ssi}}$ benchmark. 136
7.2 Measures of the Example Runs 151

A.1 Summary of constraints on IPL joint profiles. 175
A.2 Residual STDE for Linear Models 178

List of Boxes

1.1 Non-deterministic procedures. 14
1.2 Epistemological status of ML. 17
1.3 ML Design Space. 24

2.1 Mechanistic *vs.* Phenomenological Approaches 40

List of Listings

6.1 Pseudocode for $(+i)$ and $(-i)$ RNN accommodation in OSL. 108

7.1 Pseudocode for $(+i)$ and $(-i)$ accommodation in Q-Learning. 142
7.2 Pseudocode for $(+i)$ and $(-i)$ accommodation in Actor-Critic. 145