# Input Addition and Deletion in Reinforcement: Towards Learning with Structural Changes

Iago Bonnici, Abdelkader Gouaich, Fabien Michel

# Input Addition and Deletion in Reinforcement: Towards Learning with Structural Changes*

Iago Bonnici
iago.bonnici@lirmm.fr

Abdelkader Gouaïch
abdelkader.gouaich@lirmm.fr

Fabien Michel
fmichel@lirmm.fr

LIRMM, Université de Montpellier, CNRS
Montpellier, France

## ABSTRACT

Reinforcement Learning (RL) agents are commonly thought of as adaptive decision procedures. They work on input/output data streams called "states", "actions" and "rewards". Most current research about RL adaptiveness to changes works under the assumption that the streams signatures (*i.e.* arity and types of inputs and outputs) remain the same throughout the agent lifetime. As a consequence, natural situations where the signatures vary (*e.g.* when new data streams become available, or when others become obsolete) are not studied. In this paper, we relax this assumption and consider that signature changes define a new learning situation called Protean Learning (PL). When they occur, traditional RL agents become undefined, so they need to restart learning. Can better methods be developed under the PL view? To investigate this, we first construct a stream-oriented formalism to properly define PL and signature changes. Then, we run experiments in an idealized PL situation where input addition and deletion occur during the learning process. Results show that a simple PL-oriented method enables graceful adaptation of these arity changes, and is more efficient than restarting the process.

## KEYWORDS

reinforcement, transfer learning, online learning, recurrent network

**ACM Reference Format:**
Iago Bonnici, Abdelkader Gouaïch, and Fabien Michel. 2020. Input Addition and Deletion in Reinforcement: Towards Learning with Structural Changes. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

### 1.1 Protean Learning

In heuristic AI and weak AI, Artificial Agents (AAs) are mostly used as *search* tools. The AA is assigned a task that its user may not know how to tackle, and its goal is to search a wide space of possible behaviours until it finds one that makes it solve the task, at least approximately. In Unsupervised Learning (UL), the AA discovers structure in the data it feeds on. In Supervised Learning (SL) the AA feeds on examples realisations of target behaviour: *i.e.* a sequence of correct mappings input $\mapsto$ output, called a *training set*, which guides it towards acceptable approximations. In Reinforcement Learning (RL), no such mapping is available. Instead, the AA successively tries various behaviours, only guided by a continuously fed *reward* signal designed by user, suggesting whether or not it is currently doing well [12, 28]. In this context, the agent's *behaviour* is a streaming process continuously computing the next "action" to undergo (or agent's *outputs*) based on currently perceived "state" of the system (or agent's *inputs*). The simplicity of this abstract design is the reason RL broadly adapts a variety of situations like playing human games [21, 26], controlling 3D creatures [11, 19] or trading at high frequency [27]. Another reason is that UL or SL are typically used as methodological elements of RL, so any progress in either is also beneficial to RL. Recent loud success of function approximation tools like neural networks is a good example of such progress. For instance, Recurrent Neural Networks (RNNs) are particularly beneficial to RL because of their streaming nature [8, 25].

Ideal RL agents adapt changing conditions while still solving the task at hand. Succeeding in this adaptation is a well-known challenge tackled by various communities depending on the meaning of the word "changing". For instance, the domain of Concept Drift (CD) is concerned with changing environmental functions, resulting in that environmental responses to the same agent actions cannot be assumed to remain the same all along the learning process [10, 36].

In this paper, we consider that not only the environment changes, but also the *interface* between the agent and the environment. For instance, consider a RL agent embedded into a sticky roverbot whose task is to follow a user anywhere. Starting from an initial, trivial behaviour where it does not move at all, and feeding from sensory inputs only, RL makes this agent progressively learn how to coordinate its actions until it is able to follow its target. However, reaching a successful state where it is able to solve this task on a wooden floor means not that it will be able to follow the user later on a rocky ground (*environmental change*), when its rear camera will break (*input deletion -i*), when its left caterpillar will eventually get jammed (*output deletion -o*), when a new engine will be added to compensate (*output addition +o*), or when a new infrared sensor will be plugged in (*input addition +i*), even though the task remains the same. Non-hardware RL agents also face this challenge. For instance, consider a long-term high-frequency trading learner feeding from streaming statistical indicators gathered online [27]. 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 it is not considered relevant anymore by the trading community (*-i*), then precious resources like time, power, data, developers, would regularly be wasted. Instead, it must keep on trading and improving with the new available information.

*This article extends an earlier paper titled « Effects of Input Addition in Learning for Adaptive Games: Towards Learning with Structural Changes ».

*Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.* 

We refer to the list of available inputs and outputs streams as the agent *signature*. The question is: If a RL agent has been trained to optimize reward feedbacks under signature $\Delta_0$, can it adapt later different signatures $\Delta_1$, $\Delta_2$, *etc.*? The problem is that an agent defined by signature $\Delta_0$ is undefined under $\Delta_1$, so it cannot exist anymore and has to be redefined. But the intuition dictates that it should benefit from previous experience and be more efficient under $\Delta_1$, $\Delta_2$, *etc.* than a naive learner resuming from scratch. By explicitly considering these changes, we extend RL to a broader learning situation referred to as *Protean Learning* (PL). We expect PL agents to be able to keep learning no matter changes in their signature.

We trial this problem with two contributions in this work. After a succinct overview of related works (section 1.2), we first construct a formalization of RL that focuses on the PL view with non-constant input/output/feedback signatures (section 2.1). Second, we design and run an experiment addressing two first kinds of signature changes: *input addition* (+*i*) and *input deletion* (-*i*), in an idealized PL situation that can be solved with SL (section 2.2). We show that a surprisingly simple adaptation of traditional learning procedures makes the agent gracefully adapt the signature changes in this case, while still capitalizing on past experience (section 3).

## 1.2 Related work

Learning situations related to PL are known in the domain of Transfer Learning (TL). In TL, the agent has already found acceptable solutions to tasks called "source" tasks, and the challenge is to benefit from this previous "knowledge" while tackling a new "target" task. In other words, the TL agent is expected to generalize not only *within* tasks, but also *across* tasks [30]. This domain is transversal to Machine Learning as it applies both to SL [4, 32] and RL [18, 30]. In fact, we find that TL is commonly invoked in various different situations, although it is not always acknowledged which of these situations is currently being instantiated. Considering the literature, here are the various TL situations we distinguish:

**(1) posterior transfer:** The source training process is already done, it has been successful but costly. One wishes to benefit from TL to tackle a new target task more efficiently [29, 30].

**(2) subtasking:** The target task is challenging. One wishes to split it up into several easier source tasks, expecting that TL occurs from the smaller tasks to the big one, and that this process is better than direct tackling of the target [6, 9, 30].

**(3) joint learning:** Several different tasks have to be learned at once. One wishes that TL occurs from ones to the others, and speeds up the overall parallel process [13, 31].

**(4) prior transfer:** The task at hand will undergo future changes, but these changes are yet unknown. One wishes, prior to learning, to design an agent able to adapt these changes and benefit from TL from any task to the next. [24, 32]

PL, described in section 1.1, is an instance of situation **(4)**.

PL is also related to Concept Drift (CD), a situation where the environment is assumed to undergo changes while the agent learns [10, 14, 33, 34, 36]. It is also related to Continual Learning (CL) or "lifelong learning", an AI design where the agent keeps learning as it regularly faces new challenges like in situation **(4)** [24, 32, 35].

However, even though the environment function is expected to change in TL, CD, CL, the signature of the agent is widely assumed to be fixed in these related works. The field of Domain Adaptation (DA) tackles input changes (heterogeneous DA [7, 13]) or output changes (open-set DA [3]) in SL and in the transfer situation **(1)**. But PL focuses on signature changes in RL instead, and in the transfer situation **(4)**. As such, PL is not a direct instance of these domains, although it benefits from methods in these previous works.

As an instance of TL, PL is also transversal to SL and RL. In particular, the experiment presented in section 2.2 is an instance of online SL (oSL) so it benefits from PL.

# 2 MATERIAL AND METHODS

Signature changes cannot always be predicted in advance, but they fall into only a few categories like adding, removing or changing inputs, outputs or feedbacks. We expect a PL agent to be ready to face them. As a first step, we study the effect of adding or removing an input to the agent-environment interface during learning. Does it dramatically alter the process? If the PL agent adapts without being redefined, does it perform better than a naive learner resuming from scratch on a signature change? First, we provide a formal definition of the generic PL situation (section 2.1). Second, we design and conduct an experiment demonstrating the *sine qua non* relevance of PL approach at least in an idealized oSL case (section 2.2). In future works, PL will be investigated in abstract RL situation before we confront it to other kinds of changes and to real-world applications.

## 2.1 Formalization of PL

*2.1.1 Background.* RL is traditionally formalized using Markov Decision Processes (MDP) [28]. On each step, the RL agent perceives a "state" $s_t$: a random variable in $S$; and a random scalar "reward" $r_t$ in $\mathbb{R}$. The agent is then responsible to pick an "action" $a_t$ in $A$ according to the distribution given by its internal "policy" $\pi$:

$$a_t \hookrightarrow \pi(s_t,\ r_t) \tag{1}$$

The environment $E$ reacts to the agent action by determining the distribution of the next step state and reward:

$$(s_{t+1},\ r_{t+1}) \hookrightarrow E(a_t) \tag{2}$$

The discounted return $G_t$ is defined as the sum of future rewards starting from $t$:

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i \tag{3}$$

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

With this setting, the goal for a RL agent is expressed as an optimization problem over the space of all possible policies: Find the optimal policy $\pi^*$ so as to maximize the expected return value:

$$\pi^* = \arg\max_{\pi} \mathbb{E}(G_t|\pi) \tag{4}$$

*2.1.2 Overview of RL and PL as Case 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 $S$ and $A$. We construct a formalization that focuses on viewing RL as a *stream processing* situation.

A *data stream* is a value that changes in time. Inputs $i$ (loosely mapped to the concept of "states"), outputs $o$ (loosely mapped to

"actions") and feedbacks $\varphi$ (loosely mapped to "rewards") of a control agent are considered data streams. The agent itself is considered a *stream processing* unit that continuously transforms inputs into outputs *via* an internal *process* $P$ called its "behaviour" (and loosely mapped to "policy"). The objective of RL is that values of the $\varphi$ stream become and remain high.

On the other hand, the environment $E$ is another stream processing unit working the other way round. It continuously transforms the output streams $o$ into input streams $i$ and feedbacks $\varphi$, 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.

*2.1.3 Data Streams and Causality.* Streams are represented by functions of continuous time, like $g\colon \mathbb{R}^+ \to D$. They take their value in arbitrary domains D. Streams are discretized in time with arbitrary precision $\epsilon \in \mathbb{R}^{+*}$ by sequences ${}^{\epsilon}g : \mathbb{N} \to D$ such that:

$$\forall t \in \mathbb{N},\ {}^{\epsilon}g(t) = g(\epsilon\, t) \tag{5}$$

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 a *determination function* $f$ a function able to determine an outgoing stream $h$ from an incoming stream $g$ no matter the precision $\epsilon$ considered: $\forall \epsilon \in \mathbb{R}^{+*},\ \forall t \in \mathbb{N}$,

$${}^{\epsilon}h(t) = f_{\epsilon}\big({}^{\epsilon}g(0),\ \ldots,\ {}^{\epsilon}g(t-1),\ {}^{\epsilon}g(t)\big) \tag{6}$$

Note that stream determination has a *memory* in that current value of $h$ may depend on past values of $g$, so it is only called "Markovian" if it allows hidden states. It is also *causal* in that future values of $h$ cannot be determined given current and past values of $g$. We use the following graphical alias to represent determination relation (6):

$$g \text{ —— } (f) \longrightarrow h \tag{7}$$

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, $i$ —— $(P)$ ⟶ $o$ means that the inner agent process $P$ feeds from input stream $i$ to produce the output stream $o$ in a causal, maybe non-Markovian way, *i.e.* it may exhibit *memory*. Conversely, $o$ —— $(E)$ ⟶ $i$ means the environment works the other way round.

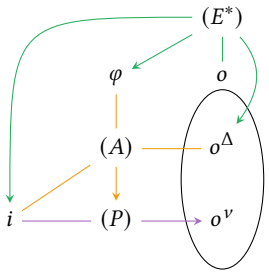
*2.1.4 Multiple Streams and Signatures.* A *multiple stream* $g$ carries both a stream of *domains* noted $g^{\Delta}$, whose values are called *signatures*, and a stream of *values* noted $g^{\nu}$ (see example Fig. 1 left). A signature is a tuple of domains $(D_1,\ D_2,\ \ldots)$ and values are elements from these domains $(v_1 \in D_1,\ v_2 \in D_2,\ \ldots)$. For instance, at $t = 0.9$, the sticky roverbot (see section 1.1) that is sensitive to both "user direction and ground speed" receives, as signature and values:

$$g^{\Delta}(0.9) = \big([0,\ 2\pi[,\ \mathbb{R}^+\big) \quad \& \quad g^{\nu}(0.9) = \Big(0.2 \text{ rad},\ 15 \text{ cm.s}^{-1}\Big)$$

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 $g^{\Delta}$ resulting in that the agent later receives values with different domain signatures, thus the term *protean*. For instance, after its front camera has been broken, and a new battery sensor has been plugged in, the sticky agent later receives "ground speed and battery level" as

$$g^{\Delta}(1.1) = \big(\mathbb{R}^+,\ [\![1,\ 5]\!]\big) \quad \& \quad g^{\nu}(1.1) = \Big(31 \text{ cm.s}^{-1},\ 4\Big).$$

*2.1.5 Dynamical System.* At the highest level, a PL learning situation is represented by 3 multiple streams $(i,\ o,\ \varphi)$ and 1 stream of determining functions $P$ with the following determination diagram:




(8)

According to (7), (6) and to boilerplate graphical aliases not developed in this paper, the diagram (8) is equivalent to the set of formal equations (9-12). They form a Dynamical System for any time precision $\epsilon \in \mathbb{R}^+$. For the sake of readability, all $\epsilon$ symbols have been dropped in the following equations:

$$(i(0), \varphi(0), o^{\Delta}(0)) = E(\varnothing) \tag{9}$$

$$\forall t \in \mathbb{N},\ P(t) = A\Big((i(0), \varphi(0), o^{\Delta}(0)), \ldots, (i(t), \varphi(t), o^{\Delta}(t))\Big) \tag{10}$$

$$\forall t \in \mathbb{N},\ o^{\nu}(t) = P(t)\Big(i(0),\ \ldots,\ i(t)\Big) \tag{11}$$

$$\forall t \in \mathbb{N}^*,\ (i(t), \varphi(t), o^{\Delta}(t)) = E\Big(o(0),\ \ldots,\ o(t-1)\Big) \tag{12}$$

Note that $P(t)$ is a determination function. Each colored equal sign or arrow corresponds to one determination triplet (7) where:

- $E$ represents the environment in which the agent is immersed. The * means that initial values of $i,\ \varphi,\ o^{\Delta}$ are determined by $E$. For the sticky roverbot, $E$ represent physics, the target user behavior, hardware and software environment all together, *i.e.* everything that is out of the decisional agent reach $A$.
- $i$ represents the agent's *inputs* or *sensors*. For the sticky bot, $i$ informs the agent of user direction and distance. Their nature changes in time as the signature $i^{\Delta}$ evolves (*e.g.* on a sensor upgrade, a sensor plug (+$i$) or a sensor break (-$i$)).
- $o$ represents the agent's *outputs* or *actuators*. In our example, $o$ controls the caterpillar engines. Their nature also changes in time as $o^{\Delta}$ evolves. Note that output values $o^{\nu}$ are determined by the agent, but the output signature $o^{\Delta}$ is determined by the environment.
- $\varphi$ represents the agent's *feedback*, *rewards* or *objectives*, a continuously fed evaluation of the actions it undertakes. For the sticky bot, $\varphi$ measures closeness to the target or the battery level. It also changes in nature as $\varphi^{\Delta}$ evolves.

The environment determines $i$, $\varphi$ and $o^{\Delta}$, so the agent cannot directly decide its input or feedback data, nor its output signature. On the inner side:

- $P$ represents the agent current *behavior*, *policy* or *program*. It is an inner computational procedure that determines current

output values based on current inputs and all past inputs. Concrete "decisions" of the agent, represented here as $o^\nu$, are produced by $P$. Note that $P$ is a stream itself, so that it evolves in time, and decisions taken by behavior $P(t)$ are not taken later by behavior $P(t + \Delta_t)$.
- $A$ is the learning procedure of the agent. It continuously adapts the behavior $P$ based on environmental information. This is where the actual behaviour search is performed (see section 1.1). Abstract "decisions" of the agent, *i.e.* strategic choices represented here as $P$, are produced by $A$.

Only two objects are not depending on time in this system: the environment $E$ and the inner agent strategy $A$. In the sense of Formal Grammars and Dynamical Systems, $E$ and $A$ embody the *rules* of the system, while its *initial state* is represented as the first production of $E$: $(i(0),\ \varphi(0),\ o^\Delta(0))$.

*2.1.6 The Objective of PL.* The classical RL problem of "maximizing rewards" [28] is reformulated in PL as:

*Given environment $E$ and a corresponding stream of feedbacks $\varphi$ with only scalar domains, find an agent procedure $A$ such that all values taken by stream $\varphi^\nu$ are Pareto-maximized.*

$A$ is considered a good learner if it optimizes the feedbacks for a whole family of environments, *i.e.* if it adapts many environments, no matter the variable nature of the signatures streams $i^\Delta$, $o^\Delta$ or $\varphi^\Delta$.

*2.1.7 Discussion.* This formalism is unconventional, and it does not yet feature the traditional probabilistic view of RL [28]. Instead, it focuses on the agent *signature* and how it changes in time, which is essential to PL.

Under this view, PL can either be considered a *special case* of RL where not only the inputs are given as states, but also their signature, (*e.g.* $s_t \in S$ with $s_t = (i^\Delta(t),\ i^\nu(t))$); or it can be considered a *generalization* of RL where the signature is no longer considered constant, (*e.g.* $s_t \in S_t$ with variable $S_t$, for instance $S_t = \prod i^\Delta(t)$).

This formalism is also compatible with future extensions of PL where a new cause for signature changes is that PL agents/diagrams *compose* each other. For instance when 1 agent splits into 2 or when 2 agents merge together: $i^\Delta(t + \mathrm{d}t) = i_1^\Delta(t)\ i_2^\Delta(t)$.

As an instance of TL, PL fits *online learning* in general, including RL but also oSL. In oSL, $i$ and $o$ carry the learning batches. $E$ contains training sets $(i,\ o^*)$ and compares each $o(t)$ to $o^*(t)$ to compute corresponding loss $\varphi(t)$. The difference with RL is that $i(t)$ does not depend on past values of $o$. Also, $\varphi(t)$ is immediately available so there is no credit assignment problem to solve for $A$ [28]. The experiment presented next assesses basic viability of PL in oSL.

## 2.2 Experiment

Preliminary to the construction of full-fledged PL agents, we design an experiment to assess their basic viability in simple cases where: (1) the environment is abstract and controlled (2) only *input addition* (+$i$) and *input deletion* (-$i$) occur, like in Fig. 1 left, and (3) the task is a simple oSL instance of PL. To this end, we restrict ourselves to an ideal situation where the optimal behavior $P^*$ is known from the experimenter. The environment $E$ is able to access a *training set* of example optimal realizations $T = \left\{\left(i_n,\ o_n^*\right)\right\}_{n\in(1,\ \ldots,\ 1000)}$ with every $i_n \text{ — } (P^*) \rightarrow o_n^*$. Note that, for the purpose of the experiment, the timeline of $i_n$ and $o_n^*$ sequences (horizontal in Fig. 1) does not correspond to the learning timeline $t$ (horizontal in Fig. 2) as it would in RL, so credit assignment does not come into play yet [28]. The agent only explores the space of behaviours to approximate $P^*$. To simulate the signature change, we stop the SL procedure, change the signature of inputs $i_n$, change the signature of $A$ and $P$ with a TL technique, then resume SL. Comparison is made with a naive, non-PL agent that directly learns from scratch with the new signature.

The following sections describe the successive steps of the experiment and its controlled parameters:

- Generate synthetic inputs $i_n$ with controlled correlation ($\kappa$) and autocorrelation ($\rho$).
- Construct ideal behaviour $P^*$ with controlled complexity ($c$).
- Compute ideal outputs $o_n^*$ with control of relevant input ($\alpha$).
- Construct learning agent $A$. Start learning $P^*$. Stop.
- Simulate signature change (+$i$ or -$i$). Resume learning.
- Measure the advantage of PL agent compared to naive agent.

*2.2.1 Inputs Generation.* Inputs carry information supposed to help the agent in its task. This information is more or less predictable. For instance with the sticky bot, the position of the target is expected not to differ much between time steps if the target is moving smoothly, but it is hard to predict if the target is fast and erratic. We expect this predictability to influence the agent reaction to a change in input signature, and assess it by generating various synthetic input data with a controlled level of autocorrelation $\rho$. When several information channels are available, they also are more or less correlated together. For instance, when an infrared camera is plugged into the sticky bot, it essentially carries information similar to the classical camera; but a battery sensor will produce 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 $\kappa$. 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, 3 reflected Gaussian random walks $i_{n_a},\ i_{n_b},\ i_{n_c}$ are independently generated in $[-1,\ 1]$ [16], with initial value uniformly chosen and standard deviation $\rho$. Then (see Fig. 1, top right), they are combined as:

$$i_n = \begin{pmatrix} i_n^1 \\ i_n^2 \end{pmatrix} = \begin{pmatrix} \kappa\, i_{n_b} + (1-\kappa)\, i_{n_a} \\ \kappa\, i_{n_b} + (1-\kappa)\, i_{n_c} \end{pmatrix} \tag{13}$$

The lower $\rho$, the more predictable $i_n$. The higher $\kappa$, the more redundant the two channels. Three values are tested for $\rho$ to assess the effect of input predictability: 0.05 (smooth), 0.15 (noisier) and 10 (almost white noise). Jointly, three values are tested for $\kappa$ to assess the effect of channels redundancy: 0 (independent channels), 0.5 (the channels carry correlated information) and 1 (same information).

*2.2.2 Optimal Outputs Generation.* The optimal behaviour used, $P^*$, is a two-steps process (see Fig. 1, bottom right):

$$i_n \text{ — } (P^{*\prime}) \longrightarrow i'_n \text{ — } (P^{*\prime\prime}) \rightarrow o_n^* \tag{14}$$

The first step $P^{*\prime}$ is to merge the two channels of $i_n$ into one:

$$i'_n = \alpha\, i_n^1 + (1-\alpha)\, i_n^2 \tag{15}$$

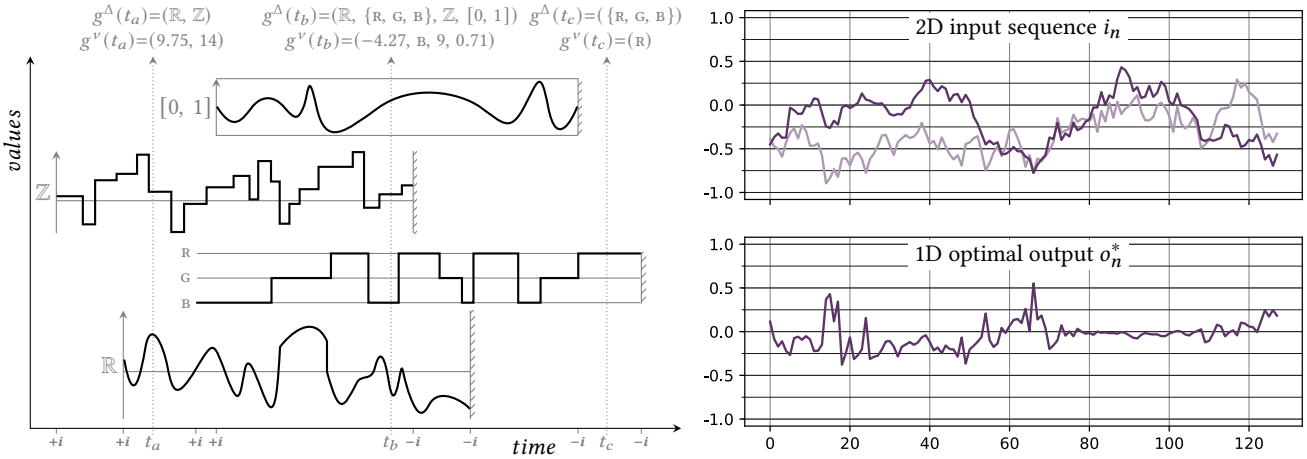



**Figure 1: Left: Example multiple stream $g$. Each curve corresponds to the temporary presence of one domain in the signature stream $g^\Delta$. If $g$ represents an agent input stream, then this learner is initially sensitive to a measure valued in $\mathbb{Z}$. It later becomes also sensitive to a measure valued in $\mathbb{R}$, then in $[0,\ 1]$. All these sensitivities are eventually lost between $t_b$ and $t_c$. However, the learner ends up 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. Right: Example synthetic streams for the preliminary experiment. Top: autocorrelated random input stream $i_n$ with 2 correlated channels. Bottom: 1-channel output stream $o_n^*$ computed with transformed random Legendre polynomial combination. In this example, $\rho = .15$ (intermediate autocorrelation level), $\kappa = .5$ (intermediate correlation level), $\alpha = .5$ (balanced usefulness of input channels), and $c = (2, 3)$ (intermediate polynomial complexity).**

with a parameter $\alpha \in [0,\ 1]$ explained hereafter. The second step is to transform $i'_n$ into $o_n^*$ with a parametrized determination function $P^{*\prime\prime}$ that permits fine control of the task complexity.

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 the way an agent reacts to changes in its input signature. To assess this, we generate tasks of various complexity by choosing that $P^{*\prime\prime}$ is constituted of a random multivariate polynomial $\mathcal{P}$. The measure of complexity $c\colon (m, d)$ is twofold. The first component $m$ is the depth of $P^{*\prime\prime}$ memory, *i.e.* how many past values of $i'_n$ are used to compute one step $\mathcal{P}(i'_n(t-m+1),\ \ldots,\ i'_n(t))$. So $m$ corresponds to the dimension of $\mathcal{P}$. The higher $m$, the more complex the task. The second component $d$ is the degree of $\mathcal{P}$. The higher $d$, the less linear $\mathcal{P}$, and the more complex the task.

Generating random polynomials with controlled degree $d$ is not straightforward, and 3 methodological obstacles need be overcome:

**1:** A polynomial of degree $d$ with random parameters may not behave as a full-degree polynomial. For instance, it can be almost linear, and the task complexity becomes over-estimated. To avoid this, $\mathcal{P}$ is defined as a random mixture of successive Legendre polynomials $(\mathcal{L}_k)_{k\in[\![0,\ d]\!]}$ [1], as each $\mathcal{L}_k$ makes full use of its degree $k$:

$$\mathcal{P}_l = \sum_{k=0}^{d} (-1)^{\sigma_k}\, w_k\, \mathcal{L}_k, \qquad \sum_{k=0}^{d} w_k = 1 \tag{16}$$

$$\mathcal{P} : \begin{cases} [-1,\ 1]^m & \to\ [-1,\ 1] \\ (x_1,\ \ldots,\ x_m) & \mapsto\ \prod_{l=1}^{m} \mathcal{P}_l(x_l) \end{cases} \tag{17}$$

The weights values $w_k$ are drawn from a Dirichlet distribution, and the random signs $\sigma_k$ are drawn from a Bernoulli distribution.

**2:** $\mathcal{P}$ is almost degenerated when the dominant weight $w_d$ is lower than the other weights, resulting in the task complexity being over-estimated. To avoid this, the Dirichlet distribution concentration is set to $(1,\ \ldots,\ 1,\ 2)$ so $w_d$ is on average twice higher.

**3:** When $m$ and $d$ increase, values of $\mathcal{P}(x)$ become biased towards 0, so $\mathcal{P}$ is easily approximated with the null function, and the task complexity is over-estimated. To avoid this, an additional transformation is applied to the values of $\mathcal{P}(x)$ so as to stretch them away from 0. The transformation aims to restore the biased distribution to $\mathcal{U}([-1,\ 1])$. To this end, we first evaluate the distribution of $\mathcal{P}(x)$: For each tested value of $m$ and $d$, 200 random polynomials $\mathcal{P}$ are drawn and evaluated in 2000 random points $x$. The distribution of all outputs $\mathcal{P}(x)$ is estimated with a Gaussian Kernel Density Estimation (KDE) (Scott bandwidth selection) [15]. We restore the distribution by using approximation of corresponding cumulative density function $CDF$ (256 linear interpolation points) as the stretching transformation. The final formula used for $P^{*\prime\prime}$ is:

$$o_n^*(t) = 2 \times CDF \circ \mathcal{P}\Big(i'_n(t),\ \ldots,\ i'_n(t-m+1)\Big) - 1 \tag{18}$$

Three values of complexity $c$ are tested. They correspond to properties of $\mathcal{P}$: (1, 1) (linear, Markovian), (2, 3) (cubic, remembers last iteration) and (4, 3) (cubic, reaches 4 steps back in time).

Not all inputs are equally useful to solve the task at hand. For instance, the battery level does not help the sticky bot when it has to follow its target. We expect this relative usefulness to influence the agent reaction to input addition or deletion. To assess it, we tune the value of $\alpha$ when mixing the 2 inputs channels of $i_n$ into $i'_n$

(equation (15)). The higher $\alpha$, the more useful the first input channel and not the other one. Three values are tested for $\alpha$: 0 ($i_n^1$ is useless to solve the task), 0.5 ($i_n^1$ is useful but not sufficient to solve the task) and 1 ($i_n^1$ contains all information needed to solve the task).

*2.2.3 Agent Structure and Learning.* The agent approximates optimal behaviour $P^*$ with actual behaviour $P$. As a fixed parameter of the experiment, we choose a Recurrent Neural Network (RNN) to implement $P$. RNNs are known to adjust arbitrarily complex recursive functions and infer arbitrarily numerous hidden states provided they contain enough internal states [8, 25]. Therefore, they model any internal representation of the agent so that it progresses towards $P^*$. As suggested by [**?** ], 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 [5], with 6 internal states each, the last one being used as the network output. $P$ produces the actual agent outputs according to $i_n \text{ — } (P) \rightarrow o_n$.

The learning procedure $A$ processes training examples by batches of 100, and updates the weights parameters of $P$ with a stochastic gradient descent and Adam update rule [17] (learning rate = 0.01). On each batch, Mean Squared Error $MSE(o,\ o^*)$ is calculated as a loss. Convergence is achieved using pytorch [22] for 1000 iterations.

*2.2.4 Realization of Signature Change.* Three convergences are achieved on each run (see Fig. 2). In the case of input addition (+$i$):

(1) One protean "first-form" agent $A_{f_1}$ (blue, left trace) is constructed with a 1D input signature. Its parameters are randomly initialized from $\mathcal{U}([-.01,\ .01])$. $A_{f_1}$ is trained against $T$ but only feeds from channel $i_n^1$ of input stream, blind to $i_n^2$. Note that in general, $P^*$ cannot be reached in this case, only its projection to the closest 1D function can be approximated.

(2) One protean "second-form" agent $A_{f_2}$ is constructed with a 2D signature. Its initial parameters are copied from the latest parameters in $A_{f_1}$, except for the necessary additional parameters that are set to zero. This simple form of TL is considered transfer of "low-level" knowledge in [30] with obvious mapping from source to target task. $A_{f_2}$ is then trained against the whole training set $T$ (green, right light trace), not ignoring channel $i_n^2$ anymore: $P^*$ can be reached.

(3) One traditional "direct" agent $A_d$ is constructed with a 2D signature. Its parameters are randomly initialized, and it is directly trained against the whole training set $T$ (black trace).

In the case of input deletion (-$i$), the protocol is reversed: $A_{f_1}$ (2D) is able to see the whole dataset (blue, left trace), and $A_{f_2}$, $A_d$ (1D) (red, black, right traces) are both blind to the second channel. Note that a few networks parameters are then lost during the transfer.

The couple $(A_{f_1}, A_{f_2})$ is our experimental oSL model of a PL agent. It experiences 2 elementary signature changes: +$i$ and -$i$. 1000 replicates are run for each combination of experimental settings, with inputs sequences 128 values long, and always with a new $P^*$.

*2.2.5 Measure of the Advantage.* Three measures are taken to assess the advantage of PL compared to direct learning. They are computed on the learning curves $l : t \mapsto MSE(o(t),\ o^*(t))$ (Fig. 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 $A_{f_1}$ and the mean first 100 of $A_{f_2}$:

$$\text{IT} = \frac{1}{100}\left( \sum_{t=-100}^{-1} \log_{10}\left(l_{A_{f_1}}(t)\right) - \sum_{t=0}^{99} \log_{10}\left(l_{A_{f_2}}(t)\right)\right) \tag{19}$$

(Note that time is counted negatively prior to the event.) 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*:

$$\text{LT} = \frac{1}{1000} \sum_{t=0}^{999} \log_2\left(\frac{l_{A_d}(t)}{l_{A_{f_2}}(t)}\right) \tag{20}$$

LT = 1 means that second-form agent is twice better than direct agent on average. LT = 0 means that they perform similarly.

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

$$\text{LP} = \frac{1}{100} \sum_{t=900}^{999} \log_{10}\left(l_{A_{f_2}}(t)\right) \tag{21}$$

When LP is below -2 (MSE<$10^{-2}$), we consider the task to be solved.

## 3 RESULTS AND DISCUSSION

The measures obtained on each run are summarized in Fig. 3. Various effects discussed here are named A, B, *etc.* They are illustrated in Fig. 2 with example runs drawn from key conditions in Fig. 3.

A generalized linear model was fitted on the data to address relevance of observed variations. Predictions are represented as dots in Fig. 3. All interactions were considered between experimental settings $\rho$, $\kappa$, $\alpha$ and $c$ considered as factors (degrees of freedom: 80919, residual stde: 0.7822). The effects discussed hereafter only rely on high significance contrasts with $p\text{-}value \leqslant .001$. Convergence and analysis of the model were achieved with R-Cran software [23].

As a general trend, even though 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 $A_{f_2}$ compared to the naive, direct agent $A_d$. In other words: performing the transfer with a PL approach is more beneficial than restarting the learning from scratch when the signature change occurs. This advantage needs be qualified depending on the situation:

A. *PL benefits from input redundancy*: In the +$i$ situation, it is expected that, when the new input carries information similar to existing one, transfer makes $A_{f_2}$ benefit from prior learning compared to $A_d$. This is confirmed by the data when $\alpha < 1$: the higher $\kappa$, the higher LT.

B. *Input loss implies 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 big difference although LT scores are still positive on average. B supports our abstract model of a task.

C. *PL 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

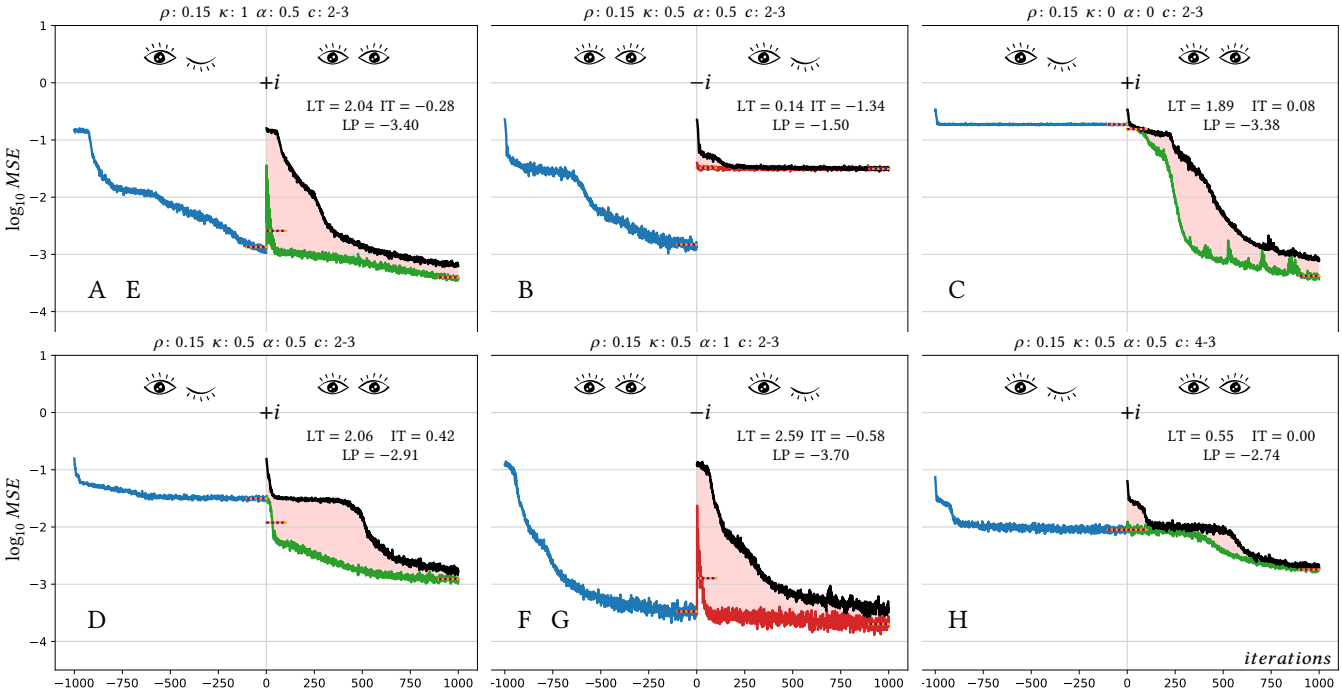



**Figure 2: Learning curves $l$: evolution of MSE (section 2.2.4) for key example individual runs in the experiment. Filled areas illustrate the LT measure (20). Dotted bars illustrate the IT measure (19) and the LP measure (21). Eyes represent 1D/2D cases.**

$\alpha = 0$ and $\kappa = 0$, and higher if $\rho$ is low. Our interpretation is that $A_{f_1}$ still learns a correct representation/preprocessing of the data in this case, and that this knowledge benefits to $A_{f_2}$.

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

E. *There are negative transfer perturbation*: Effect D does not hold in -$i$ when $\alpha = 1$ or $\kappa = 1$: IT scores are negative. *I.e.* 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.

F. *PL recovers with redundancy*: In -$i$, the removal perturbation is expected to be less severe when the inputs are redundant. This is confirmed by the data when $\alpha < 1$: the higher $\kappa$, the higher IT (note that IT $< 0$, still). $A_{f_2}$ 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. *PL suffers from redundancy*: The effect F is reversed when $\alpha = 1$: the higher $\kappa$, 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 $A_{f_2}$ to figure out that the information lost is still available in the remaining input. Note that the long-term LT scores are still positive.

H. *Complexity levels it off*: It is expected and observed that, the more complex the task ($c$), the less intense all the effects listed above. Indeed, when both $A_{f_2}$ and $A_d$ struggle to lower the error, their relative advantage becomes less clear. Still, LT remains positive on average.

As a summary, we observe that PL is overall beneficial on the long-term after a signature change. Most of these results were expected, but it is worth noting that a short negative transfer perturbation (IT$< 0$) is observed in special cases (input removal, redundant new input), and that there were no guarantees that this effect would not overwhelm the whole learning process and take over the long-term scores LT during the experiment. We have thus demonstrated that transfer works as expected in PL for input addition/deletion signature change events, at least in this idealized oSL situation, and that the protean approach of online learning is still conceivable.

A few effects were unexpected. C shows that transfer is beneficial even if the prior agent was completely unable to solve the task at hand. We suppose it learns information about structure of the data it is later fed from, and we suggest that this phenomenon be studied as an alternative parameter initialization procedure. A parsimonious hypothesis is that it only learns to *ignore* the useless input.

According to TL theory [30], there can be various reasons for the advantage of the second-form agent $A_{f_2}$: First, $A_{f_2}$ initial loss can be lower than $A_d$ because $A_{f_1}$ has already started converging; this is known as the *jumpstart* benefit. Second, $A_{f_2}$ loss can decrease faster than $A_d$: $A_{f_2}$ is said to *learn faster*. Lastly, $A_{f_2}$ final loss can be lower than $A_d$: $A_{f_2}$ is said to *learn better*. The LT metric (20) is an aggregated estimation of these 3 possible advantages, and IT (19) only measures jumpstart. As such, we cannot distinguish all effects from each other. However, considering that many runs exhibit negative IT yet positive LT, we can assert that the jumpstart effect

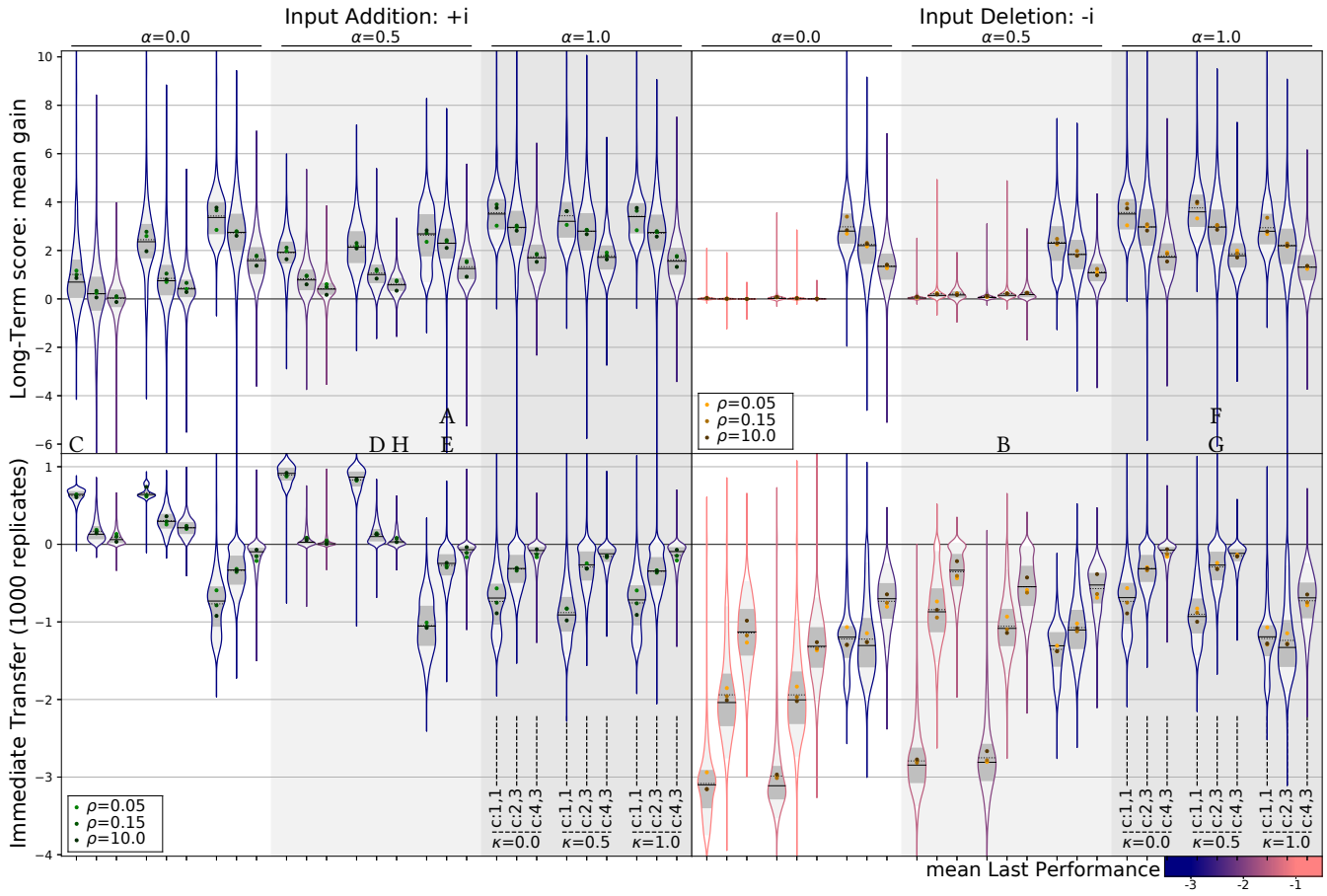



**Figure 3: Violin plot: Comparison of metrics in the various experimental settings. Left panes: input addition. Right panes: input deletion. Top panes: Long-Term advantage LT as defined in (20). Bottom panes: Immediate Transfer IT as defined in (19). Violin border color indicates the mean value of the Last Performance LP as defined in (21). Violins are aggregated over $\rho$ to ease readability, but statistical predictions of the metrics depending on $\rho$ are represented as dots within the violins. Solid lines represent median values, dashed line represent mean values. Grey areas in the violins represent 50% and 90% percentiles.**

is not the only benefit of PL in this experiment. This was an open question in [2], and the effect C is another argument in this favor.

The experiment presented here is idealized and abstract. It assesses the *sine qua non* condition that PL learners can be developed at least in an easy situation like oSL. Extending these preliminary results to RL is therefore a matter of confronting to Incremental Learning [20] and the Credit Assignment problem [28], and is the subject of current work in progress. *I.e.*, the learning/optimization timeline (Fig. 2) must line up with the data stream timeline (Fig. 1).

## CONCLUSION

RL agents are expected to continuously adapt their environments. Real-world situations challenge them with changes in their input/output stream signature. In this paper, we have generalized the idea of RL to a broader PL learning situation that explicitly takes these signature changes into account. First, we have presented a novel formal vision of RL making it possible to extend to PL. Then, with a controlled oSL experiment, we have started exploring how PL learners can be developed. We have shown that low-level transfer techniques correctly address input addition and input deletion when learning non-Markovian and non-linear tasks, at least with RNNs in the idealized case, and we have determined various detailed effects in play during the process. These preliminary results are encouraging, as they suggest that control agents can learn even if their agent-environment interface is changing. In subsequent works, consistently with our general approach, PL will be used again to address other types of signature changes like output/feedback addition and removal ($\pm o, \pm \varphi$) or agents composition/decomposition. Traditional RL benchmarks like mountain car and cart pole balancing [28] or Atari Games [21, 26] and 3D creatures [11, 19] cannot be directly used in evaluating PL as they do not feature signature changes yet. We need adapting them in order to compare PL to common RL approaches. In the future, the above results will also need to confront full-fledged generic tasks, real-world problems and software applications.

## REFERENCES


[1] Milton Abramowitz and Irene A. Stegun. 1972. Lengendre Functions. In *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables* (9th ed.). New York: Dover, 771–802.

[2] Iago Bonnici, Abdelkader Gouaïch, and Fabien Michel. 2019. Effects of Input Addition in Learning for Adaptive Games: Towards Learning with Structural Changes. In *EvoApplications: Applications of Evolutionary Computation*, Vol. LNCS. Leipzig, Germany, 172–184. https://doi.org/10.1007/978-3-030-16692-2_12

[3] Pau Panareda Busto and Juergen Gall. 2017. Open Set Domain Adaptation. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, 754–763. https://doi.org/10.1109/ICCV.2017.88

[4] Rich Caruana. 1994. Learning Many Related Tasks at the Same Time with Backpropagation. In *Proceedings of the 7th International Conference on Neural Information Processing Systems (NIPS'94)*. MIT Press, Denver, Colorado, 657–664.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, Yoshua Bengio, and Dzmitry Bahdanau. 2014. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014).

[6] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. 2016. Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer. *CoRR* abs/1609.07088 (2016).

[7] Lixin Duan, Dong Xu, and Ivor W. Tsang. 2012. Learning with Augmented Features for Heterogeneous Domain Adaptation. *CoRR* abs/1206.4660 (2012).

[8] J Elman. 1990. Finding Structure in Time. *Cognitive Science* 14, 2 (June 1990), 179–211. https://doi.org/10.1016/0364-0213(90)90002-E

[9] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. 2017. Meta Learning Shared Hierarchies. *CoRR* abs/1710.09767 (2017).

[10] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *Comput. Surveys* 46, 4 (March 2014), 1–37. https://doi.org/10.1145/2523813

[11] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous Deep Q-Learning with Model-Based Acceleration. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 2829–2838.

[12] Christopher J. Hanna, Raymond J. Hickey, Darryl K. Charles, and Michaela M. Black. 2010. Modular Reinforcement Learning Architectures for Artificially Intelligent Agents in Complex Game Environments. In *Computational Intelligence and Games*. IEEE, Copenhagen, Denmark, 380–387. https://doi.org/10.1109/ITW.2010.5593329

[13] Maayan Harel and Shie Mannor. 2010. Learning from Multiple Outlooks. *CoRR* abs/1005.0027 (2010).

[14] Heng Wang and Zubin Abraham. 2015. Concept Drift Detection for Streaming Data. In *International Joint Conference on Neural Networks*. IEEE, Killarney, Ireland, 1–9. https://doi.org/10.1109/IJCNN.2015.7280398

[15] M. C. Jones, J. S. Marron, and S. J. Sheather. 1996. A Brief Survey of Bandwidth Selection for Density Estimation. *J. Amer. Statist. Assoc.* 91, 433 (March 1996), 401–407. https://doi.org/10.1080/01621459.1996.10476701

[16] Tahır A. Khanıev, İhsan Unver, and Selahattın Maden. 2001. On the Semi-Markovian Random Walk with Two Reflecting Barriers. *Stochastic Analysis and Applications* 19, 5 (Oct. 2001), 799–819. https://doi.org/10.1081/SAP-120000222

[17] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[18] Alessandro Lazaric. 2012. Transfer in Reinforcement Learning: A Framework and a Survey. In *Reinforcement Learning*, Marco Wiering and Martijn van Otterlo (Eds.). Vol. 12. Springer, 143–173.

[19] Timothy P. Lillicrap, Jonathan J. Hunt, Alexand er Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous Control with Deep Reinforcement Learning. *arXiv e-prints* (Sept. 2015), arXiv:1509.02971.

[20] Viktor Losing, Barbara Hammer, and Heiko Wersing. 2018. Incremental On-Line Learning: A Review and Comparison of State of the Art Algorithms. *Neurocomputing* 275 (Jan. 2018), 1261–1274. https://doi.org/10.1016/j.neucom.2017.06.084

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (Feb. 2015), 529–533. https://doi.org/10.1038/nature14236

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. (2017).

[23] R Core Team. 2018. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[24] Mark Bishop Ring. 1994. *Continual Learning in Reinforcement Environments*. Ph.D. Dissertation. University of Texas at Austin, Austin, TX, USA.

[25] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (Jan. 2015), 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

[26] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529 (2016), 484–503.

[27] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market Making via Reinforcement Learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '18)*. International Foundation for Autonomous Agents and Multiagent Systems, Stockholm, Sweden, 434–442.

[28] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press, Cambridge, Mass.

[29] F. Tanaka and M. Yamamura. 2003. 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. IEEE, Kobe, Japan, 1108–1113. https://doi.org/10.1109/CIRA.2003.1222152

[30] Matthew E. Taylor and Peter Stone. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10, 7 (2009), 1633–1685.

[31] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. 2017. Distral: Robust Multitask Reinforcement Learning. *CoRR* abs/1707.04175 (2017).

[32] Sebastian Thrun. 1995. Is Learning the N-Th Thing Any Easier Than Learning the First?. In *Proceedings of the 8th International Conference on Neural Information Processing Systems (NIPS'95)*. MIT Press, Denver, Colorado, 640–646.

[33] Alexey Tsymbal. 2004. The Problem of Concept Drift: Definitions and Related Work. (2004).

[34] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning* 23, 1 (April 1996), 69–101. https://doi.org/10.1007/BF00116900

[35] Ju Xu and Zhanxing Zhu. 2018. Reinforced Continual Learning. *CoRR* abs/1805.12369 (2018).

[36] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. 2016. An Overview of Concept Drift Applications. In *Big Data Analysis: New Algorithms for a New Society*, Nathalie Japkowicz and Jerzy Stefanowski (Eds.). Vol. 16. Springer International Publishing, Cham, 91–114. https://doi.org/10.1007/978-3-319-26989-4_4