Input Addition and Deletion in Reinforcement: Towards Protean Learning

Iago Bonnici, Abdelkader Gouaich, Fabien Michel

JAAMAS manuscript No. AGNT-D-20-00064
https://doi.org/10.1007/s10458-021-09534-6

Input Addition and Deletion in Reinforcement: Towards Protean Learning

Iago Bonnici · Abdelkader Gouaïch · Fabien Michel

Received: 30 June 2020 / Accepted: 15 September 2021

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 networks

1 Introduction

Artificial AI agents can be considered *search* tools. The agent is first assigned a task that its user may not know how to tackle. Then, the agent goal is to search a wide space of possible *behaviours* until it finds one that solves the task, at least approximately. This particular

I. Bonnici
Orcid 0000-0003-2934-351X
LIRMM, Univ Montpellier, CNRS
Montpellier 34095 Cedex 5 France
E-mail: iago.bonnici@lirmm.fr

A. Gouaïch
Orcid 0000-0003-2537-2321
E-mail: gouaich@lirmm.fr

F. Michel
E-mail: fmichel@lirmm.fr

approach comes in several flavours, commonly brought together under the generic term "Machine Learning" (ML).

In Unsupervised Learning (UL), the agent feeds on raw data, and the outcome of its *behaviour* is a structuring of this data. The expectation of UL is that interesting hidden structure is found by the UL agent this way (clusters, correlations, distribution..) [19]. One challenge of UL is to choose the right amount of prior knowledge to inject in the agent, so it is aware of the patterns to look for, but not biased towards one in particular.

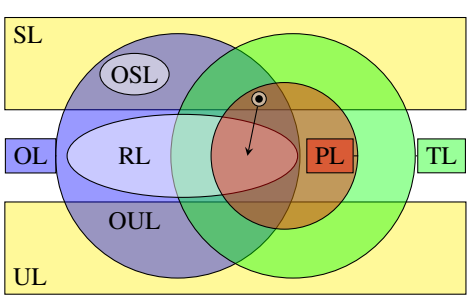
In Supervised Learning (SL), there is one target behaviour, and the agent feeds on sample realisations of this behaviour. Typically, the samples take the form of a sequence of correct mappings (*input*, *output*) called a *training set*. The agent's own *behaviour* is a function whose realisations mimic the training set. And the expectation of SL is that this function eventually converges towards an acceptable approximation of the target behaviour [30]. One challenge of SL is to make the agent's search procedure flexible enough so it can approach any target behaviour, but without fitting the training set so close that no generalization can be done, a problem known as *overfitting*.

In Reinforcement Learning (RL), no training set is available. Instead, the agent is embedded into an environment whose dynamics are unknown, and it explores various possible behaviours until it eventually improves on the task. Its only guide is a *reward* signal, designed by the user, used to indicate the situations where it is doing well [40, 16]. In this context, the agent's *behaviour* is a streaming process endlessly computing the next "action" to undergo (or agent's *outputs*) based on currently perceived "state" of the environment (or agent's *inputs*), while the environment retroactively updates inputs the other way round. One major challenge in RL is that the environment introduces delays in the feedback loop. As a consequence, the agent never determines for sure which past action to reinforce when a positive reward is received. This is known as the Credit Assignment problem.

Despite this intrinsic challenge, RL design is sufficiently generic to yield efficient results in a variety of situations like playing human games [37, 29], controlling 3D creatures [14, 27] or trading algorithms [39]. Another reason for this success is that UL or SL are commonly used as technological elements of RL, so any progress in either is also beneficial to RL. For instance, recent rise of function approximation tools like neural networks (NNs), and Recurrent Neural Networks (RNNs) in particular, is especially beneficial to RL because of their flexible streaming nature [10, 36].

Unlike basic SL or UL, RL happens *online*, which means that the input data stream and the agent search proceed on the same timeline. Put it another way, before every search step, the agent needs to wait for the next piece of input data to be available, so the search procedure needs to be *incremental*. We refer to this situation as Online Learning (OL) [35]. OL is transversal to ML, so it is not restricted to RL, and there exists forms of Online UL or Online SL (OSL) [35]. In OSL for instance, instead of accessing the whole training set at once, the agent receives a stream of elementary training (*input*, *output*) pairs, and produces a stream of candidate behaviours on the fly until it finds one that satisfies the user. OSL can therefore be considered a degenerated subclass of RL, where the agent data feed just happens to not be influenced by its own activity.

The learning situation we introduce in this paper aims at broadening RL to situations where the input/output data streams of the agents vary in structure during the behavioural search. The paper targets RL, but it uses OSL as a preliminary experimental model of RL (Fig. 1).



SL: Supervised Learning.
UL: Unsupervised Learning.
OL: Online Learning.
OSL: Online Supervised Learning.
OUL: Online Unsupervised Learning.
RL: Reinforcement Learning.
TL: Transfer Learning.
PL: Protean Learning.
⊙ : the presented experiment.
→ : the roadmap.

Fig. 1: Informal overview of ML fields as a Venn Diagram, dedicated to position PL in the landscape. PL is a particular case of TL, as it focuses on signature changes across tasks (Section 1.2). Like TL, PL intersects with OL when the task happens to change during the course of learning. As such, PL is also an extension of RL. The formalism presented in Section 3 suits the whole intersection of PL with OL. As the shades suggest, we defend that OSL can be considered a particular case of RL. For instance, the experiment presented in Section 4 is a case of PL in OSL, which will be later extended to PL in generic RL.

1.1 Protean Learning

Ideal RL agents adapt to changing conditions while still solving the task at hand. Succeeding in this adaptation is a well-known challenge tackled by various OL communities depending on the meaning of the word "changing". For instance, the domain of Concept Drift (CD) is concerned with changing distributions of OSL input data. Regarding RL, CD deals with changing environmental functions. A changing environment is difficult to tackle because its responses to the same agent actions cannot be assumed to remain the same all along the learning process [12, 53].

In this paper, we consider that not only the environment, as a function, 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. Eventually, the agent reaches a successful state where it manages this task on a wooden floor. But there are still a few things it is likely unable to do, like:

- Following the target on a rocky ground, *i.e.* accommodating *environmental change*,
- Adapting its behaviour when its rear camera eventually breaks, or when its left caterpillar gets jammed, *i.e.* dealing with *input deletion* or *output deletion*,
- Improving when a new engine is added to compensate, or a new infrared sensor is plugged in, *i.e.* exploiting *output addition* or *input addition*,

Note that the task still remains the same.

Facing these new kinds of changes would not only be useful in modular robotics, as the situation arises in every learning context where the data streams processed by the agent are transient because their relevance or availability varies over time. System resilience would for instance be improved on sensor upgrades or failures in autonomous vehicles. Non-hardware RL agents also face this challenge. For instance, consider a long-term automatic trading learner feeding from streaming statistical indicators gathered online [39]. Should this agent

be erased and restart the learning from scratch whenever a new indicator is created or when an old indicator is disregarded because it is not considered relevant anymore by the trading community, then precious resources like time, power, data, developers, would regularly be wasted. Instead, the agent must keep on working and improving with the new available information.

We refer to the list of available inputs and outputs streams as the agent *signature*. Our main question is: If a RL agent has been trained to optimize feedbacks under signature Δ_0, can it gracefully adapt to new signatures Δ_1, Δ_2, *etc.*? The problem is that an agent defined by signature Δ_0 is undefined under Δ_1, so it cannot exist anymore and has to be redefined. On the other hand, intuition dictates that it should benefit from previous experience and be more efficient under Δ_1, Δ_2, *etc.* than a naive agent resuming the learning from scratch. And the benefit should even be stronger when every new signature is close to the previous one. By explicitly considering these changes, we extend RL to a broader learning situation referred to as *Protean Learning* (PL). We expect PL agents to keep learning no matter the changes in their signature.

We trial this problem with two contributions in this work. First, we construct a formalization of RL, that focuses on the PL view with non-constant input / output / feedback signatures. Second, we design and run an experiment addressing two first kinds of signature changes: *input addition* and *input deletion*, in an idealized PL situation where only OSL is involved instead of full-fledged RL (Fig. 1). 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.

After an overview of related works (Section 1.2) and of the roadmap to PL (Section 2), we present the formalization (Section 3) and the experiment (Section 4) before we expose and discuss the results (Section 5) and we conclude along with the next works to undertake.

1.2 Related Works

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, and the objective is to benefit from this prior "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 [42,45]. This domain is transversal to ML as it applies to UL [50,7], SL [44,5] and RL [42,25]. The core motivation of TL is to improve learning efficiency on the target task. Depending on the situation, this improvement takes various forms, and it can take several:

- Either the *initial performance* of the agent on the target task is better than the initial performance of a naive agent starting from scratch. This is known as the *jumpstart* benefit, and it happens because the convergence process has already started during source learning.
- Or the agent *learns faster* on the target task, because it inherits from some form of intimacy with the source task. It therefore reaches a successful state sooner.
- Or the *final performance* of the agent on the target task is better than the performance of a naive agent. This happens for instance if dead-end local optima have already been avoided and ruled out during source learning.

A common pitfall in TL is that there are very few theoretical guarantees that the target learning process will converge towards acceptable behaviours. In addition, two negative

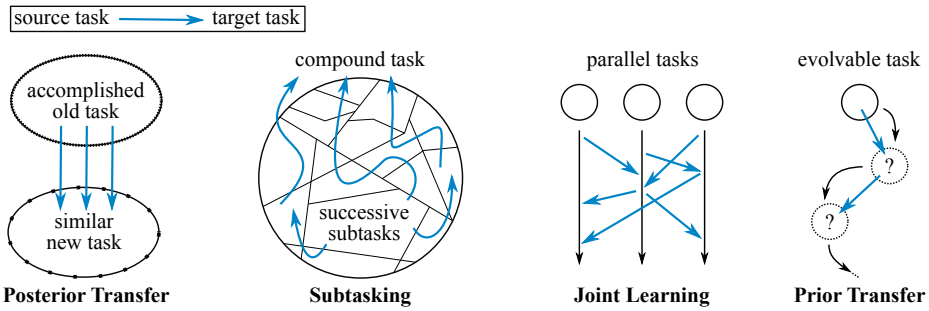


Fig. 2: Various different learning situations where Transfer Learning is invoked.

effects possibly occur: First, when an agent succeeds in solving the target task, it is sometimes not able to solve the source task anymore. This phenomenon is known as *Catastrophic Forgetting*, and is undesirable when the source task is expected to be faced again [24,34, 52,21]. Second, it sometimes happens that an agent performing the transfer benefits from none of the above improvements, or that it ends up *less* efficient on the target task than a naive learner. This phenomenon is known as *Negative Transfer*, and there are very few theoretical guarantees against it [42,45,13]. As an instance of TL, PL also inherits from *Catastrophic Forgetting* and *Negative Transfer* pitfalls and from the tradeoff between two extreme approaches. In the approach we refer to as *conservative*, agents give up resources like storage memory and exploration steps to ensure that performance in earlier contexts does not decrease. On the other hand, *progressive* agents allocate all resources to improve current and future performances regardless of past contexts. Conservative agents are advantaged in situations were earlier contexts are likely to occur again, for instance if the agent is training at the base and broken sensors can eventually be fixed. Progressive agents are advantaged in critical situations were no reversion is possible, for instance on Mars or in critical radiation zones. This work explores PL from the progressive perspective and *Negative Transfer* is the only concern addressed in Sections 4 and 5.

Regarding the literature, we find that there are various different situations in which TL is invoked, but that it is not always acknowledged which of these situations is currently being instantiated. Since we consider that one of these situations includes PL, but not the others, it is useful that we distinguish at least 4 different use cases for TL (Fig. 2):

(1) Posterior Transfer: In this situation, a source training process has already been done, and it has been successful but costly. Now that a new target task has to be tackled, one wishes to benefit from this prior experience to improve efficiency on the target. In short, one wishes that transfer occurs from the old task to the new task [41,42].

(2) Subtasking: In this situation, there is one challenging target task to tackle. 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, then to successively combine them together to ease the learning path towards the whole target. In short, one wishes that transfer occurs from the small tasks to the big task [42,9,11].

(3) Joint Learning: In this situation, several tasks have to be learned at once. In order to improve efficiency of the the overall parallel process, it is expected that any progress made in one task is immediately propagated to the other tasks so they can benefit from

it without needing to discover it by themselves. In short, one wishes that transfer occurs from the ones to the others [5,43,17].

(4) Prior Transfer: In this situation, the task at hand 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, one wishes that transfer occurs from any task to the next [33,44].

PL, as described in Section 1.1, is an instance of Prior Transfer **(4)**, because we consider that signature changes, in the general case, are unpredictable events that PL agents are expected to face.

PL is also related to Concept Drift (CD). In CD, the environment is assumed to undergo changes while the agent is still learning, and numerous approaches aim at classifying, detecting, and adapting to these changes [48,46,12,18,53]. Like Prior Transfer, in principle, PL is also closely related to Continual Learning (CL) or "lifelong learning", a more general AI design where the agent is expected to keep learning forever while it regularly faces new challenges. In other words, even the task is expected to change [44,33,49,52,21,38].

However, even though the task or the environment function is expected to change in TL, CD and CL, the signature of the agent is widely assumed to be fixed in these related works. The notion of signature is examined in the field of Domain Adaptation (DA). For instance, heterogeneous DA is concerned with input changes [26,17], and open-set DA can be considered an attempt to perform output changes [4]. However, they both tackle SL in the implicit Posterior Transfer situation **(1)**, whereas PL focuses on RL in Prior Transfer **(4)**. As such, PL is not a direct instance of these domains, although it does benefit from methods developed in these previous works.

2 Roadmap to PL

This section states our overall approach of PL, sketching a coarse-grained, high-level roadmap towards ideal full-fledged PL. Signature changes like the ones we described in Section 1.1 cannot always be predicted in advance, but they fall into only a few categories. The various different ways a PL agent signature can evolve are listed below. For convenience, each type of change is also referred to with a short symbol like (+i) for *input addition*, (-o) for *output deletion*, or ($\sim\phi$) for *feedback change*:

- The agent benefits from an *addition* of some data slots: inputs (+i), outputs (+o) or feedbacks (+ϕ). For instance, this happens when new sensors or actuators are plugged into the sticky roverbot, or when new parallel objectives are defined by the user, like watching the battery level in addition to following the target.
- The agent suffers a *removal* of some data slots: inputs (-i), outputs (-o) or feedbacks (-ϕ). For instance, this happens as sensors and actuators break, or as they become obsolete and the user removes them and cancels objectives. When data becomes uninformative because of an environmental change, like a blurry camera input due to fog or heavy rain, then the information is lost but the (uninformative) data is still produced so the agent signature is the same. In this situation, there is no (-i) event but ($\sim E$), unless a higher-level procedure, or a meta-agent, be able to detect the change and decide to explicitly silent the camera instead (-i).

- There is a *change* in the domains of possible values for some data slots: inputs ($\sim i$), outputs ($\sim o$) or feedbacks ($\sim \phi$). For instance, this happens when a temperature sensor widens its sensitivity range, or when a wheel motor is upgraded to also feature backwards spinning. From a learning perspective, there are at least two ways of handling this situation:
  - Either the set of possible values for the data slots is actually updated. For instance, if the agent's possible output values for the wheel motor were initially ranging in [0: stop, 0.5: half-speed, 1: full speed], then after the ($\sim o$) event, they would range in [-1: full speed backwards, 0: stop, 1:full speed]. This explicitly states to the agent that the range $[-1, 0[$ has not been explored yet, and that new environmental responses are expected if these values are tested.
  - Or 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, if the agent's possible output values for the wheel motor were initially ranging in [0: stop, 0.5: half-speed, 1: full speed], then after the ($\sim o$) event, they would now mean [0: full speed backwards, 0.5: stop, 1:full speed]. This approach is easier to tackle as it reduces any change event ($\sim i$), ($\sim o$) or ($\sim \phi$) to a simple environmental change ($\sim E$). As such, it does not actually involve PL and can be tackled with CD techniques, but it is not equivalent to the former one because it is less informative for the agent.
- The agent *splits* into separate pieces, so the data slots are separated 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 with a smaller signature.
- Several agents *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, which augments its signature.

CD-aware 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 ($+i$), ($+o$), ($+\phi$), ($-i$), ($-o$), ($-\phi$), ($\sim i$), ($\sim o$), ($\sim \phi$), (*split*) and (*merge*) events, without being undefined, and without throwing away their previous experience. An ideal PL agent is able to adapt any of these and keeps learning no matter the changes in its signature.

Obviously this is a challenging goal, and the involved problems differ much depending on the actual event involved. For instance, one challenge with ($+i$) and ($+o$) is to trigger the exploration of possible behaviours again, especially if the agent has already satisfyingly converged [15]. It 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 the agent performance decreases if it was heavily relying on the lost data slots, so another challenge is to gracefully ease the regression. It is also the case with ($\sim i$) and ($\sim o$) when previously possible values become impossible. Regarding ($+\phi$) and ($-\phi$), 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 [51], so one challenge in this case is for the user to correctly specify their precise expectations. In contrast, a simple ($\sim \phi$) event can be considered a subset

of ($\sim E$), which is already the object of active research in the fields of CD, CL and TL (Section 1.2).

The involved challenges also depend much on the learning methods actually applied. For instance, even though NNs 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 objectives like *splitting* agents extremely challenging with today's state of the art, but see [47] for an insight into modularizing NNs. On the other hand, *merging* NN-based agents together does not imply that their knowledge be understood or modular, but it suggests that a combinatory explosion of possible interactions between all data slots and inner states of the networks be managed, which is another critical challenge to be met.

As a first step towards PL, we construct a formalization of this learning situation to embrace the above speculated types of signature change (Section 3). Then, we conduct and design an experiment demonstrating the *sine qua non* viability of PL at least in an idealized case, with a typical RNN-based OSL agent undergoing (+i) and (-i) change events during the course of its learning (Section 4). The following questions are addressed: Do the change events dramatically alter the learning process? If the PL agent adapts without being redefined, does it perform better than a naive learner resuming from scratch on a signature change? After we answer these, future works will relax the ideal assumptions in the experiment, and meet other kinds of signature changes like (+o), (-o) *etc.*.

3 Formalization

3.1 Background

RL is traditionally formalized using Markov Decision Processes (MDP) [40]. On each step, 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 the set of possible actions $\mathcal{A}$, according to the probability distribution given by its current internal behaviour called "policy" π:

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

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

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

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

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

The discount factor $\gamma \in [0, 1[$ represents a recency principle, resulting in that rewards in the near future are better valued than subsequent rewards. With γ, the sum always converges.

With this setting, the behavioural search for a RL agent is expressed as an optimization problem over the space of all possible policies: Find the optimal policy $\hat{\pi}$ that maximizes the expected return value G:

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

3.2 PL as 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 $\mathcal{S}$ and $\mathcal{A}$. We construct a formalization that focuses on viewing RL as a *stream processing* situation.

A *data stream* is a value that changes over time. Inputs i (loosely mapped to the concept of "states"), outputs o (loosely mapped to "actions") and feedbacks ϕ (loosely mapped to "rewards") of a control agent are considered as data streams. The agent itself is considered a *stream processing* unit that endlessly transforms inputs i into outputs o *via* an internal *process* P called its "behaviour" (and loosely mapped to "policy"). The objective of RL is that values in the ϕ stream become and remain high.

Outside the agent scope, 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 ϕ, embodying the dynamics of the system the agent is embedded within.

The *signature* describes the interface between the agent and the environment, including 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.

3.3 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 $\varepsilon \in \mathbb{R}^{+*}$ by sequences ${}^{\varepsilon}g : \mathbb{N} \to D$ such that:

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

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 ε considered: $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$,

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

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. Stream determination is also *causal* in that only the current value of h, but not its future values, can be determined given only current and past values of g. We use the following graphical alias to represent the determination relation (6):

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

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, $i \text{ —} (P) \text{→ } o$ means that the inner agent process P feeds from input stream i to produce the output stream o in a causal, maybe non-Markovian way, *i.e.* it may exhibit *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 typically ends up in a recursive definition, so it is ill-defined. To express the agent-environment retroaction loop, we therefore need to bootstrap their dynamics with another kind of relation, $\forall \varepsilon \in \mathbb{R}^{+*}$:

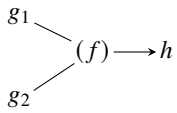
$$\begin{cases} {}^{\varepsilon}h(0) = f_{\varepsilon}(\varnothing) \\ \forall t \in \mathbb{N}^*,\ {}^{\varepsilon}h(t) = f_{\varepsilon}({}^{\varepsilon}g(0),\ \ldots,\ {}^{\varepsilon}g(t-1)) \end{cases} \tag{8}$$

The relation (8) introduces a one-step delay determination function. And the determination function is also able to determine the first consequence value $h(0)$ on its own. This is graphically represented as:

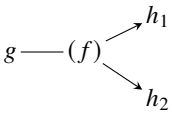
$$g \text{ —}(f^*)\text{→ } h \tag{9}$$

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

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



(10)



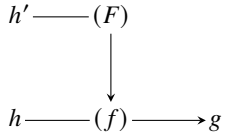
(11)

Similarly to (6-7), the above two diagrams respectively translate as, $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$:

$${}^{\varepsilon}h(t) = {}^{\varepsilon}f\Big(\big({}^{\varepsilon}g_1(0),\ {}^{\varepsilon}g_2(0)\big),\ \ldots,\ \big({}^{\varepsilon}g_1(t),\ {}^{\varepsilon}g_2(t)\big)\Big) \tag{12}$$

$$\big({}^{\varepsilon}h_1(t),\ {}^{\varepsilon}h_2(t)\big) = {}^{\varepsilon}f\big({}^{\varepsilon}g(0),\ \ldots,\ {}^{\varepsilon}g(t)\big) \tag{13}$$

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



(14)

The latter diagram being equivalent to, $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$:

$$\begin{cases} {}^{\varepsilon}f(t) = {}^{\varepsilon}F({}^{\varepsilon}h'(0),\ \ldots,\ {}^{\varepsilon}h'(t)) \\ {}^{\varepsilon}g(t) = {}^{\varepsilon}f(t)\big({}^{\varepsilon}h(0),\ \ldots,\ {}^{\varepsilon}h(t)\big) \end{cases} \tag{15}$$

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

3.4 Multiple Streams and Signatures

Data streams with variable arity and variable types are represented with particular streams that we call *multiple streams*. A multiple stream $g = g^{\Delta,\nu}$ carries both a stream of *domains* noted g^{Δ}, whose values are called *signatures*, and a stream of *values* noted g^{ν} (Fig. 3 left). A signature is a tuple of domains $(D_1,\ D_2,\ \ldots)$ and values are elements from these domains

$(\nu_1 \in D_1,\ \nu_2 \in D_2,\ \ldots)$. For instance, at $t = 0.9$, the sticky roverbot (Section 1.1) that is sensitive to both "direction to user" and "ground speed" receives the signature and values:

$$g(t) = g^{\Delta,\nu}(t) = \begin{pmatrix} g^{\Delta}(t) \\ g^{\nu}(t) \end{pmatrix} = \begin{pmatrix} ([0,\ 2\pi],\ \mathbb{R}^{+}) \\ (0.2 \text{ rad},\ 15 \text{ cm.s}^{-1}) \end{pmatrix} \tag{16}$$

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^{Δ} resulting in that the agent later receives values with different domain signatures, thus the term *protean*. For instance, at $t + \delta t = 1.1$, 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

$$g(t+\delta t) = g^{\Delta,\nu}(t+\delta t) = \begin{pmatrix} g^{\Delta}(t+\delta t) \\ g^{\nu}(t+\delta t) \end{pmatrix} = \begin{pmatrix} (\mathbb{R}^{+},\ [\![1,\ 5]\!]) \\ (31 \text{ cm.s}^{-1},\ 4) \end{pmatrix} \tag{17}$$

It is possible that the signature stream g^{Δ} and the values stream g^{ν} are caused by two different determination functions. In this situation, we write the following:

$$h \text{—} (f) \text{—} \overset{g}{\overset{\rightarrow}{g^{\Delta}} \quad \overset{\leftarrow}{g^{\nu}}} \text{—} (f') \text{—} h' \tag{18}$$

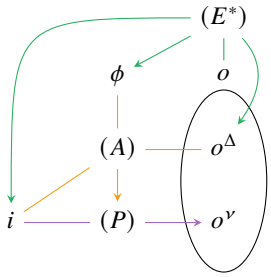
And consistently with (6-7), the above is an alias for, $\forall \varepsilon \in \mathbb{R}^{+*}$, $\forall t \in \mathbb{N}$:

$$\begin{cases} {}^{\varepsilon}g^{\Delta}(t) = {}^{\varepsilon}f\left({}^{\varepsilon}h(0),\ \ldots,\ {}^{\varepsilon}h(t)\right) \\ {}^{\varepsilon}g^{\nu}(t) = {}^{\varepsilon}f'\left({}^{\varepsilon}h'(0),\ \ldots,\ {}^{\varepsilon}h'(t)\right) \end{cases} \tag{19}$$

Note that a consistency constraint must hold for this to be well-defined: at any time t, every value in $g^{\nu}(t)$ must belong to the corresponding domain in $g^{\Delta}(t)$.

3.5 Learning Dynamics

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



(20)

According to (6-19), the diagram (20) is equivalent to the set of formal equations (21–24). They form a dynamical system for any time precision $\varepsilon \in \mathbb{R}^{+*}$. For the sake of readability, all

ε symbols have been dropped in the following equations:

$$\begin{cases} \Big(i(0),\ \phi(0),\ o^{\Delta}(0)\Big) = E(\varnothing) & (21) \\ \forall t \in \mathbb{N},\ P(t) = A\Big(\Big(i(0),\ \phi(0),\ o^{\Delta}(0)\Big),\ \ldots,\ \Big(i(t),\ \phi(t),\ o^{\Delta}(t)\Big)\Big) & (22) \\ \forall t \in \mathbb{N},\ o^{\nu}(t) = P(t)\Big(i(0),\ \ldots,\ i(t)\Big) & (23) \\ \forall t \in \mathbb{N}^*,\ \Big(i(t),\ \phi(t),\ o^{\Delta}(t)\Big) = E\Big(o^{\Delta,\nu}(0),\ \ldots,\ o^{\Delta,\nu}(t-1)\Big) & (24) \end{cases}$$

Note that $P(t)$ is a determination function itself. Each colored equal sign or arrow corresponds to one determination triplet like (7) or (9), where:

- E represents the environment in which the agent is immersed. Initial values of i, ϕ and o^{Δ} 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^{Δ} 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^{Δ} evolves. Note that output values o^{ν} are determined by the agent, but the output signature o^{Δ} is determined by the environment.
- ϕ represents the agent's *feedback*, *rewards* or *objectives*, a continuously fed evaluation of its current behaviour. For the sticky bot, ϕ measures closeness to the target or the battery level. It also changes in nature as ϕ^{Δ} evolves.

The environment determines i, ϕ and o^{Δ}, so the agent does not 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^{ν}, 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 (Section 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: $(i(0),\ \phi(0),\ o^{\Delta}(0))$.

3.6 The Objective of PL

Like in RL, the objective of PL is to optimize the values yielded by the reward stream ϕ [40], but the protean nature of ϕ make this less straightforward to formulate.

First, not every domain is suitable for the signature of the reward stream ϕ^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 $\{\text{R}, \text{G}, \text{B}\}$, which can be used as regular inputs i, cannot be used as feedback values unless they are ordered.

Second, the reward stream ϕ is a multiple stream, so there exists, in the general case, a wide range of non-Pareto-dominant "optimal" behaviours to search for a PL agent. In addition, marginal optimality is not well defined with respect to only one discounted temporal stream of feedbacks, because PL data streams cannot be assumed to last forever. As such, the classical RL problem of "maximizing rewards" needs to be reformulated in PL as:

Given an environment E and a corresponding multiple stream of feedbacks ϕ with only ordered domains in its signature ϕ^Δ, find an agent procedure A such that all values taken by the values stream ϕ^ν are Pareto-optimized.

In addition, A is considered a good learner if it optimizes the feedbacks for a whole family of environments $\mathcal{E}$ with $E \in \mathcal{E}$, *i.e.* if it adapts many environments, no matter the variable nature of the signatures streams i^Δ, o^Δ or ϕ^Δ.

3.7 Discussion

The formalism presented here is novel. It departs from the traditional view of of RL in that it does not yeat feature the probabilistic aspects of reinforcement [40]. 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 or a *generalization* of RL:

- Under the former perspective, not only the inputs are given as states, but also their signature, *e.g.* $s_t \in \mathcal{S}$ with $s_t = (i^\Delta(t),\ i^\nu(t))$.
- Under the latter perspective, the signature is no longer considered constant, *e.g.* $s_t \in \mathcal{S}(t)$ with variable $\mathcal{S}$, for instance $\mathcal{S}(t) = \prod i^\Delta(t)$.

We expect this formalism to be also useful when signature changes are the result of agent *composition*. For instance, in future extensions of PL (Section 2), we expect that one agent can split into two or two agents merge into one, and their inputs be separated or joined, *e.g.* $i^\Delta(t + \mathrm{d}t) = i_1^\Delta(t)\ i_2^\Delta(t)$.

As an instance of Prior Transfer (Section 1.2), PL fits OL in general, including RL but also OSL (Fig. 1). In OSL, the data streams i and o carry the successive learning batches. The environment E contains the training set information $(i,\ \hat{o})$, *i.e.* samples of optimal behaviour, and the agent A produces each $o(t)$ as an attempt to mimic the received $\hat{o}(t)$. E compares the stream o with $\hat{o}$ to compute the corresponding *loss* stream ϕ and feeds it back to A. The difference with RL is that $i(t)$ does not depend on past values of o. Also, $\phi(t)$ is immediately available after $o(t)$ has been emitted. In other words, E is always Markovian with no hidden state and there is no Credit Assignment problem [40]. OSL is therefore easier than RL, and the experiment presented hereafter assesses basic viability of PL in OSL.

4 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 (Fig. 3 left), and 3. the task is a simple OSL instance of PL (Section 3.7).

To this end, we restrict ourselves to an ideal situation where the optimal behavior $\hat{P}$ is known from the experimenter. The environment E is able to access a *training set* of example optimal realizations: $T = \{(i_n, \hat{o}_n)\}_{n\in(1,\ ...,\ 1000)}$, with every $i_n \text{—}(\hat{P})\rightarrow \hat{o}_n$ according to equations (6-7). Note that, for the purpose of the experiment, the timeline of i_n and $\hat{o}_n$ sequences (horizontal axes in Fig. 3) does not correspond to the learning timeline t (horizontal axes in Fig. 4) as it would in RL, so Credit Assignment does not come into play yet [40]. The agent only explores the space of possible behaviours in search for an approximation of $\hat{P}$. To simulate the signature change, we stop the SL procedure during this search, we change the signature of inputs i_n, we change the signature of A and P with a TL technique, then we resume SL. Comparison is made with a baseline, naive, non-PL agent that directly learns from scratch with the new signature, so it does not benefit from transfer. We measure whether the PL agent learns better than the naive one after the signature change event.

When (+i) or (-i) occurs, the behavioural search space needs to be redefined to contain programs with correct arity. In extended works presented in [2], we show that there exists a set of *natural projections* able to cast behaviours from one search space to the next in a generic fashion. The transfer techniques addressed in this experiment are two instances of natural projections.

The experiment happens in multiple steps, with controlled parameters ρ, κ, α and c:

1. Generate synthetic inputs i_n with controlled correlation (κ) and noise (ρ).
2. Construct ideal behaviour $\hat{P}$ with controlled complexity (c).
3. Compute ideal outputs $\hat{o}_n$ with control of relevant input (α).
4. Construct learning agent A and make it start learning so P approximates $\hat{P}$.
5. Pause learning to simulate the signature change ((+i) or (-i)), then resume learning.
6. Measure the advantage of PL agent compared to naive agent.

All steps are described in the following sections.

4.1 Inputs Generation

Inputs i carry information supposed to help the agent in its task. This information is more or less predictable. For instance, with the sticky bot described in Section 1.1, the position of the target is expected not to differ much between time steps if the target moves smoothly, but it is hard to predict if the target is fast and erratic. As this predictability influences learning, we expect that it also influences the agent reaction to a change in input signature. We assess it by generating various synthetic input data with a 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 sticky bot, the information it carries is essentially 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 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 $i_n^a,\ i_n^b,\ i_n^c$ [22] are independently generated in the range $[-1,\ 1]$, with initial value uniformly chosen and standard deviation ρ. Then (Fig. 3,

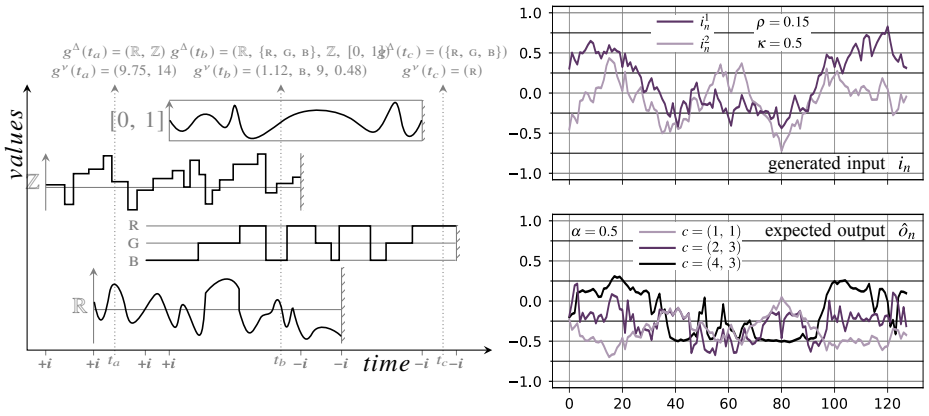


Fig. 3: **Left:** Example multiple stream $g = g^{\Delta,\nu}$. Each curve corresponds to the temporary presence of one domain in the signature stream g^{Δ}. If g represents an agent input stream, then this learner is initially sensitive to a measure valued in $\mathbb{Z}$. The learner later becomes 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 $\hat{o}_n$ computed from i_n with transformed random Legendre polynomial combination, 3 example levels of complexity.

top right), they are combined as:

$$i_n = \begin{pmatrix} i_n^1 \\ i_n^2 \end{pmatrix} = \begin{pmatrix} \kappa\, i_n^b + (1-\kappa)\, i_n^a \\ \kappa\, i_n^b + (1-\kappa)\, i_n^c \end{pmatrix} \tag{25}$$

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

4.2 Optimal Outputs Generation

The optimal behaviour used, $\hat{P}$, is a two-steps process (Fig. 3, bottom right):

$$i_n \text{ —}(Q)\text{→ } i'_n \text{ —}(R)\text{→ } \hat{o}_n \tag{26}$$

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

$$i'_n = \alpha\, i_n^1 + (1-\alpha)\, i_n^2 \tag{27}$$

with a parameter $\alpha \in [0, 1]$ explained hereafter. The second step is to transform i'_n into $\hat{o}_n$ with a parametrized determination function R 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 with various complexity by choosing that R is constituted of a random multivariate polynomial $\mathcal{P}$. The measure of complexity $c : (m, d)$ is twofold:

- The first component m is the depth of R 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 multivariate polynomials with controlled degree d is not straightforward, and three methodological obstacles (**1**, **2**, **3**) need to 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:

$$\mathcal{P}_l = \sum_{k=0}^{d} (-1)^{\sigma_k}\, w_k\, \mathcal{L}_k, \quad \sum_{k=0}^{d} w_k = 1 \tag{28}$$

$$\mathcal{P} : \begin{cases} [-1,\ 1]^m & \to\ [-1,\ 1] \\ (x_1,\ \ldots,\ x_m) & \mapsto\ \prod_{l=1}^{m} \mathcal{P}_l(x_l) \end{cases} \tag{29}$$

The weights values w_k are drawn from a Dirichlet distribution, and the random signs σ_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 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 $\mathcal{P}(x)$ become biased towards 0, so $\mathcal{P}$ 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 $\mathcal{P}(x)$ 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) with Scott bandwidth selection [20]. 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:

$$\hat{o}_n(t) = 2 \times CDF \circ \mathcal{P}\Big(i'_n(t),\ \ldots,\ i'_n(t-m+1)\Big) - 1 \tag{30}$$

Three values of complexity c are tested, which 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 only has to follow its target. We expect this relative usefulness to influence the agent reaction to input addition or deletion. To assess this, we tune the value of α when mixing the 2 inputs channels of i_n into i'_n in equation (27). The higher α, the more useful the first input channel and not the other one. Three values are tested for α: 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).

In [2], we unveil 10 possible configurations of the behavioural search space in the vicinity of (+i) and (-i) events called protean *profiles*. The two parameters α and κ in this experiment are especially important because they span 3 important profiles. When both $\alpha = 0$ and $\kappa = 0$, there is no learning challenge for the 1D agent because no information useful to the task is available, and the profile is referred to as "flat-in" or FHH in [2]. When either $\alpha = 1$ or $\kappa = 1$, the additional input is either redundant or irrelevant, so useless, and the profile is referred to as "low-out" or HLF. In other cases, the profile is the most generic or HHH. The experiment presented here therefore addresses 3 different profiles.

4.3 Agent Structure and Learning

The agent approximates optimal behaviour $\hat{P}$ with its actual behaviour P. As a fixed parameter of the experiment, we choose a Recurrent Neural Network to implement P. RNNs are known to adjust arbitrarily complex recursive functions and infer arbitrarily numerous hidden states provided they contain enough internal states [10,36]. Therefore, they model any internal representation of the agent so that it progresses towards $\hat{P}$. As suggested by our preliminary study of (+i) event [3], not having 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 [6], with 6 internal states each, the last one being used as the network output. P produces the actual agent outputs according to $i_n \text{ — } (P) \longrightarrow o_n$.

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 [23] (learning rate = 0.01). On each batch, the Mean Squared Error $MSE(o,\ \hat{o})$ is calculated as a loss. Convergence is achieved using pytorch [31] for 1000 iterations. All corresponding scripts and the produced data can be found at `https://seafile.lirmm.fr/d/0f479d1e194242a4ab2d/`.

4.4 Realization of Signature Change

Three convergences are achieved on each run (Fig. 4). In the case of input addition (+i):

1. One protean "first-form" agent A_{f_1} (left trace, in blue) 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 the input stream, being blind to i_n^2. Note that in general, $\hat{P}$ cannot be reached in this case, because the agent does not have enough information, and the signature of P even differs from $\hat{P}$. Only a *projection* of $\hat{P}$ to the closest 1D determination function can be approximated.
2. One protean "second-form" agent A_{f_2} is then 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 [42], with obvious mapping from source to target task. This constitutes a *natural projection*, because the agent starts by ignoring the additional input, so its behaviour is unchanged right after the event [2]. A_{f_2} is trained against the whole training set T (green, right light trace), not ignoring channel i_n^2 anymore, so $\hat{P}$ can eventually be reached.
3. One naive "direct" agent A_d is constructed with a 2D signature. Its parameters are randomly initialized from $\mathcal{U}([-.01, .01])$, 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. To accommodate this change, the network parameters supposed to process the second input dimension are dropped during the transfer. This cartesian reduction of the parameters constitutes an *almost-natural* projection, because it does not alter the behaviour of a degenerated agent already ignoring this channel [2].

The couple $\left(A_{f_1},\ A_{f_2}\right)$ is our experimental OSL model of a PL agent, while the couple $\left(A_{f_1},\ A_d\right)$ represents a naive, non-PL agent. They both experience two elementary signature changes: (+i) and (-i), and their performances are compared. 1000 replicates are run for each combination of experimental settings, with inputs sequences 128 values long, and always with a freshly generated $\hat{P}$.

4.5 Measure the Advantage of PL

Three measures are taken to assess the advantage of PL compared to direct learning. They are computed on the learning curves $l : t \mapsto MSE(o(t),\ \hat{o}(t))$ (Fig. 4):

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} (time is counted negatively prior to the event):

$$\mathrm{IT} = \frac{1}{100}\left(\sum_{t=-100}^{-1} \log_{10}\left(l_{A_{f_1}}(t)\right) - \sum_{t=0}^{99} \log_{10}\left(l_{A_{f_2}}(t)\right)\right) \tag{31}$$

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*:

$$\mathrm{LT} = \frac{1}{1000}\sum_{t=0}^{999} \log_2\left(\frac{l_{A_d}(t)}{l_{A_{f_2}}(t)}\right) \tag{32}$$

LT = 1 means that second-form agent A_{f_2} is twice better than the direct agent A_d on average. LT = 0 means that they perform similarly.

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

$$\mathrm{LP} = \frac{1}{100}\sum_{t=900}^{999} \log_{10}\left(l_{A_{f_2}}(t)\right) \tag{33}$$

When LP is below -2 (MSE<10^{-2}), we consider that the task has been solved, like in all examples in Fig. 4 except B.

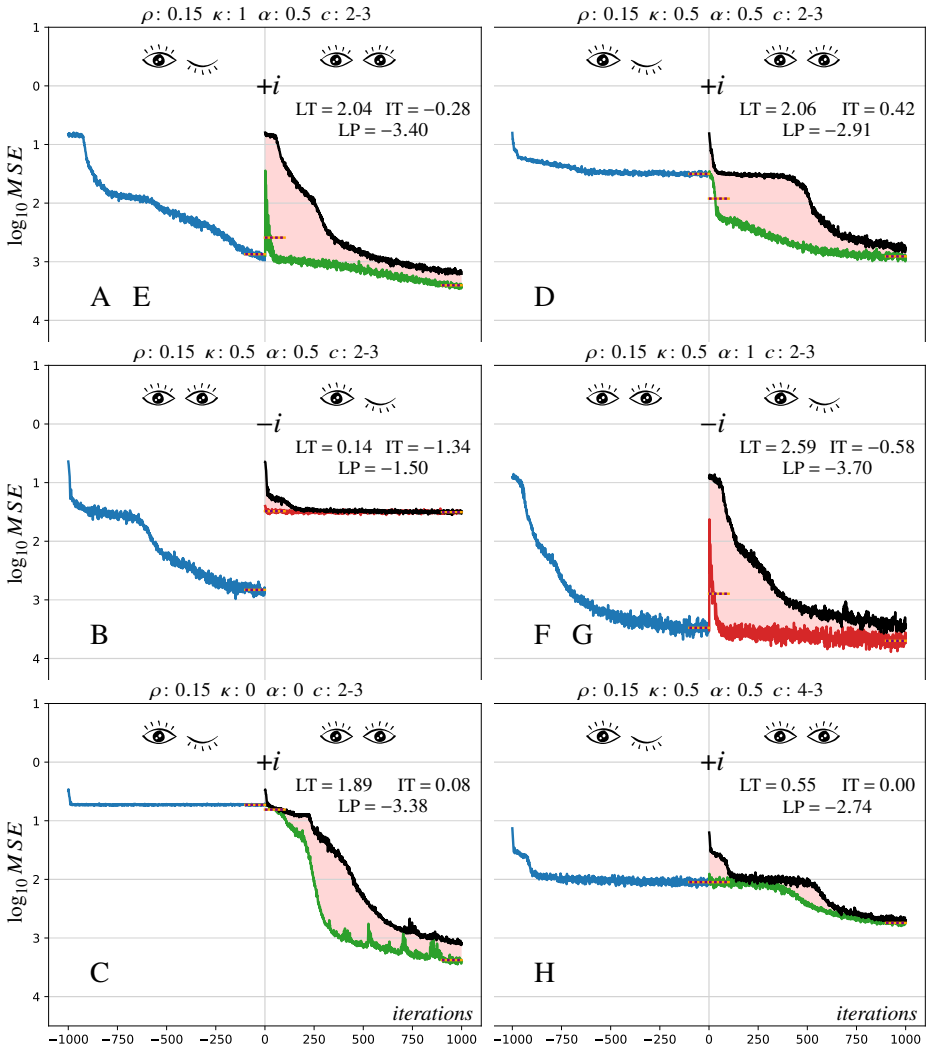


Fig. 4: Learning curves l: evolution of MSE (Section 4.4) for key example individual runs in the experiment illustrating the observed effects A, B, C, D, E, F, G and H (Section 5). Filled areas illustrate the LT measure (32). Dotted bars illustrate the IT measure (31) and the LP measure (33). The eyes represent 1D/2D situations. The signature change event occurs at time 0.

5 Results and Discussion

The measures obtained on each run are summarized in Figs. 5-6. The various effects discussed here are named A, B, C, D, E, F, G and H, and are illustrated in Fig. 4 with example runs drawn from key conditions in Figs. 5-6. The precise values represented in the figures are gathered in Tables 1, 2 and 3.

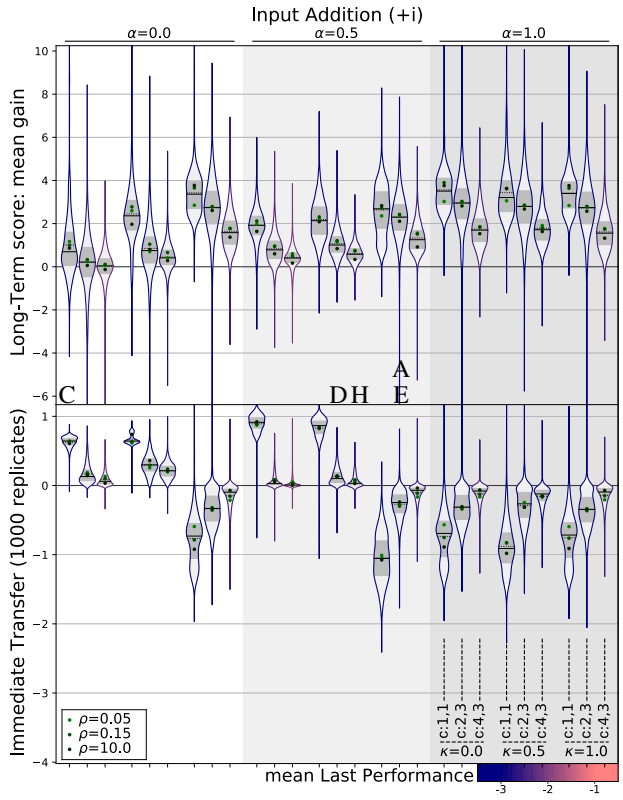


Fig. 5: Violin plot: Comparison of metrics in the various experimental settings in the case of *input addition*. Top pane: Long-Term advantage LT as defined in (32). Bottom pane: Immediate Transfer IT as defined in (31). Violin border shade indicates the mean value of the Last Performance LP as defined in (33). Violins are aggregated over ρ to ease readability, but statistical predictions of the metrics depending on ρ 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. Precise mean and standard deviation values have been gathered in Tables 1, 2 and 3.

A generalized linear model was fitted on the data to address relevance of observed variations. Predictions are represented as dots in Figs. 5-6. All interactions were considered between experimental settings ρ, κ, α and c considered as factors (degrees of freedom: 80919, residual stde: 0.7822). The effects discussed hereafter only rely on high significance contrasts with $p\text{-}value \leq .001$. Convergence and analysis of the model were achieved with R-Cran software [32]. Interaction contrasts and their significance were calculated with package phia [8].

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

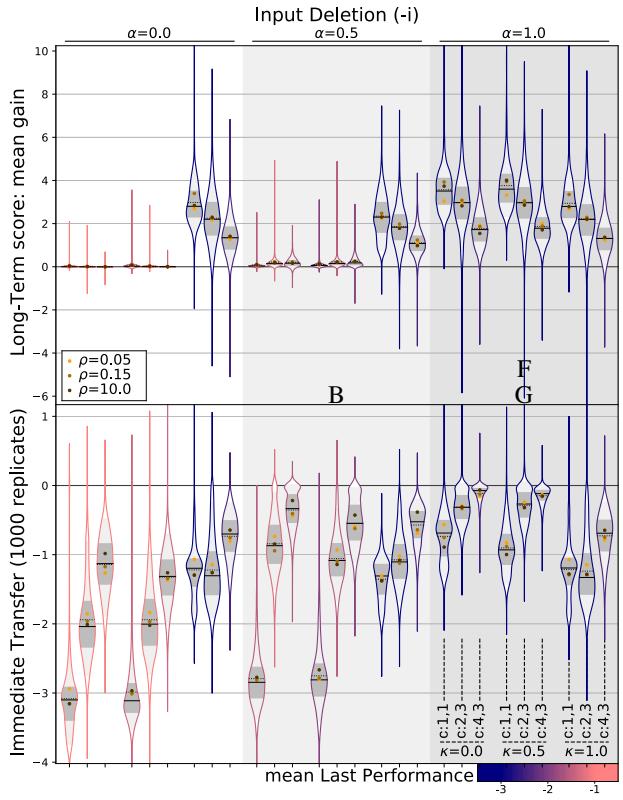


Fig. 6: Same as Fig. 5 for the *input deletion* case.

with a PL approach 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. *PL 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_{f_2} benefit from prior learning compared to A_d. This is confirmed by the data when $\alpha < 1$: The higher κ, the higher LT.
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 when the agent has not enough information.
C. *PL benefits from data structure*: In (+*i*), it is expected that transfer provides no advantage if the initial input carries no relevant information. But surprisingly, LT scores *are* positive even when $\alpha = 0$ and $\kappa = 0$, and higher if ρ 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}.

$\alpha = 0$		$(+i)$			$(-i)$		
$\kappa = 0$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	1.167 (1.381)	0.643 (0.083)	-4.175 (0.526)	0.033 (0.129)	-2.939 (0.378)	-0.924 (0.481)
	$\rho = 0.15$	0.999 (1.575)	0.634 (0.080)	-4.278 (0.512)	0.033 (0.130)	-3.154 (0.507)	-0.925 (0.481)
	$\rho = 10.0$	0.863 (1.539)	0.607 (0.095)	-4.262 (0.361)	0.035 (0.131)	-3.155 (0.601)	-0.925 (0.482)
$c = (2, 3)$	$\rho = 0.05$	0.250 (1.404)	0.187 (0.147)	-2.628 (0.705)	0.009 (0.092)	-1.852 (0.608)	-0.739 (0.347)
	$\rho = 0.15$	0.330 (1.481)	0.172 (0.133)	-2.740 (0.651)	0.011 (0.084)	-1.958 (0.591)	-0.716 (0.336)
	$\rho = 10.0$	0.061 (1.413)	0.144 (0.124)	-2.647 (0.573)	0.010 (0.080)	-2.009 (0.524)	-0.664 (0.320)
$c = (4, 3)$	$\rho = 0.05$	0.072 (0.913)	0.133 (0.105)	-2.360 (0.499)	-0.001 (0.035)	-1.266 (0.440)	-1.091 (0.282)
	$\rho = 0.15$	0.110 (0.743)	0.100 (0.087)	-2.275 (0.409)	-0.001 (0.020)	-1.174 (0.409)	-1.056 (0.248)
	$\rho = 10.0$	0.128 (0.579)	0.033 (0.053)	-1.929 (0.308)	-0.001 (0.004)	-0.984 (0.392)	-1.022 (0.230)
$\kappa = 0.5$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	2.598 (1.280)	0.615 (0.055)	-5.076 (0.415)	0.066 (0.231)	-2.985 (0.451)	-1.345 (0.483)
	$\rho = 0.15$	2.775 (1.196)	0.631 (0.054)	-5.010 (0.413)	0.067 (0.231)	-3.012 (0.502)	-1.346 (0.483)
	$\rho = 10.0$	1.970 (1.788)	0.739 (0.079)	-4.638 (0.432)	0.077 (0.235)	-2.968 (0.573)	-1.348 (0.484)
$c = (2, 3)$	$\rho = 0.05$	0.700 (1.208)	0.256 (0.144)	-2.959 (0.645)	0.025 (0.134)	-1.833 (0.579)	-0.975 (0.402)
	$\rho = 0.15$	1.046 (1.398)	0.294 (0.135)	-3.155 (0.627)	0.027 (0.133)	-1.967 (0.564)	-0.964 (0.397)
	$\rho = 10.0$	0.802 (1.316)	0.361 (0.141)	-3.020 (0.589)	0.028 (0.131)	-2.021 (0.483)	-0.892 (0.389)
$c = (4, 3)$	$\rho = 0.05$	0.410 (0.782)	0.198 (0.114)	-2.847 (0.453)	0.007 (0.039)	-1.367 (0.446)	-1.404 (0.392)
	$\rho = 0.15$	0.661 (0.662)	0.238 (0.099)	-2.877 (0.376)	0.007 (0.013)	-1.344 (0.393)	-1.377 (0.379)
	$\rho = 10.0$	0.284 (0.551)	0.211 (0.117)	-2.524 (0.325)	0.004 (0.016)	-1.262 (0.366)	-1.298 (0.386)
$\kappa = 1$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	2.851 (0.916)	-0.593 (0.204)	-4.499 (0.415)	2.696 (0.891)	-1.068 (0.257)	-4.356 (0.421)
	$\rho = 0.15$	3.658 (0.990)	-0.786 (0.340)	-4.761 (0.302)	3.399 (1.054)	-1.297 (0.417)	-4.575 (0.337)
	$\rho = 10.0$	3.771 (0.959)	-0.922 (0.443)	-4.676 (0.295)	2.838 (1.130)	-1.294 (0.482)	-4.424 (0.335)
$c = (2, 3)$	$\rho = 0.05$	2.793 (1.473)	-0.318 (0.275)	-3.088 (0.599)	2.168 (1.403)	-1.145 (0.525)	-2.958 (0.576)
	$\rho = 0.15$	2.780 (1.467)	-0.356 (0.280)	-3.160 (0.558)	2.311 (1.349)	-1.261 (0.525)	-3.058 (0.514)
	$\rho = 10.0$	2.610 (1.354)	-0.333 (0.229)	-3.062 (0.516)	2.271 (1.253)	-1.259 (0.418)	-2.990 (0.475)
$c = (4, 3)$	$\rho = 0.05$	1.783 (1.019)	-0.214 (0.204)	-2.683 (0.438)	1.270 (0.998)	-0.804 (0.381)	-2.589 (0.446)
	$\rho = 0.15$	1.766 (0.831)	-0.154 (0.158)	-2.527 (0.379)	1.381 (0.849)	-0.754 (0.332)	-2.479 (0.381)
	$\rho = 10.0$	1.372 (0.726)	-0.071 (0.094)	-2.203 (0.322)	1.414 (0.771)	-0.645 (0.250)	-2.221 (0.332)

Table 1: Mean and standard deviation values observed in the various tested experimental conditions for the three measures LT, IT and LP (eqs. (32), (31) and (33)) and illustrated in Figs. 5-6.

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 on average.
E. *There are negative transfer perturbations*: Effect D does not hold in $(-i)$ when $\alpha = 1$ 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. *PL 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_{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 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_{f_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 (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.

$\alpha = 0.5$		$(+i)$			$(-i)$		
$\kappa = 0$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	1.978 (0.570)	0.875 (0.160)	-4.751 (0.321)	0.055 (0.123)	-2.788 (0.388)	-1.521 (0.476)
	$\rho = 0.15$	2.118 (0.639)	0.886 (0.158)	-4.740 (0.320)	0.064 (0.151)	-2.814 (0.399)	-1.524 (0.478)
	$\rho = 10.0$	1.639 (1.104)	0.920 (0.160)	-4.558 (0.334)	0.091 (0.192)	-2.773 (0.496)	-1.526 (0.479)
$c = (2, 3)$	$\rho = 0.05$	0.962 (0.778)	0.081 (0.115)	-2.292 (0.463)	0.233 (0.342)	-0.736 (0.419)	-1.323 (0.326)
	$\rho = 0.15$	0.940 (0.616)	0.083 (0.117)	-2.429 (0.385)	0.223 (0.320)	-0.944 (0.377)	-1.306 (0.317)
	$\rho = 10.0$	0.601 (0.728)	0.054 (0.099)	-2.152 (0.464)	0.183 (0.299)	-0.846 (0.426)	-1.253 (0.296)
$c = (4, 3)$	$\rho = 0.05$	0.607 (0.530)	0.049 (0.079)	-2.194 (0.294)	0.215 (0.208)	-0.437 (0.289)	-1.653 (0.270)
	$\rho = 0.15$	0.491 (0.377)	0.034 (0.061)	-2.064 (0.240)	0.230 (0.198)	-0.407 (0.250)	-1.620 (0.236)
	$\rho = 10.0$	0.170 (0.470)	0.003 (0.027)	-1.670 (0.205)	0.161 (0.251)	-0.218 (0.228)	-1.551 (0.218)
$\kappa = 0.5$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	2.187 (0.838)	0.822 (0.193)	-5.253 (0.328)	0.081 (0.158)	-2.805 (0.431)	-1.942 (0.474)
	$\rho = 0.15$	2.307 (0.942)	0.826 (0.190)	-5.271 (0.335)	0.093 (0.192)	-2.783 (0.452)	-1.945 (0.478)
	$\rho = 10.0$	2.086 (1.076)	0.841 (0.191)	-5.016 (0.408)	0.129 (0.240)	-2.665 (0.515)	-1.949 (0.479)
$c = (2, 3)$	$\rho = 0.05$	1.150 (0.636)	0.126 (0.126)	-2.745 (0.381)	0.227 (0.344)	-0.932 (0.419)	-1.559 (0.374)
	$\rho = 0.15$	1.212 (0.595)	0.142 (0.130)	-2.883 (0.345)	0.231 (0.364)	-1.096 (0.386)	-1.554 (0.375)
	$\rho = 10.0$	0.839 (0.585)	0.126 (0.120)	-2.719 (0.325)	0.216 (0.322)	-1.142 (0.335)	-1.484 (0.365)
$c = (4, 3)$	$\rho = 0.05$	0.760 (0.468)	0.080 (0.089)	-2.702 (0.273)	0.242 (0.253)	-0.589 (0.357)	-1.968 (0.379)
	$\rho = 0.15$	0.729 (0.355)	0.079 (0.092)	-2.671 (0.236)	0.255 (0.231)	-0.623 (0.324)	-1.948 (0.361)
	$\rho = 10.0$	0.340 (0.350)	0.029 (0.075)	-2.245 (0.245)	0.238 (0.260)	-0.428 (0.308)	-1.856 (0.357)
$\kappa = 1$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	2.358 (0.951)	-1.009 (0.369)	-5.198 (0.644)	2.249 (0.890)	-1.307 (0.287)	-5.067 (0.681)
	$\rho = 0.15$	2.717 (1.213)	-1.058 (0.326)	-5.300 (0.522)	2.479 (1.152)	-1.379 (0.302)	-5.130 (0.595)
	$\rho = 10.0$	2.831 (1.265)	-1.076 (0.357)	-5.236 (0.568)	2.304 (0.977)	-1.377 (0.344)	-5.004 (0.656)
$c = (2, 3)$	$\rho = 0.05$	2.380 (1.135)	-0.277 (0.235)	-3.159 (0.462)	1.795 (1.105)	-1.020 (0.432)	-3.045 (0.476)
	$\rho = 0.15$	2.426 (1.191)	-0.298 (0.222)	-3.197 (0.435)	1.979 (1.030)	-1.121 (0.420)	-3.130 (0.386)
	$\rho = 10.0$	2.107 (0.925)	-0.239 (0.153)	-2.976 (0.349)	1.783 (0.856)	-1.081 (0.271)	-2.924 (0.317)
$c = (4, 3)$	$\rho = 0.05$	1.559 (0.846)	-0.166 (0.161)	-2.795 (0.364)	1.099 (0.798)	-0.688 (0.322)	-2.734 (0.347)
	$\rho = 0.15$	1.519 (0.599)	-0.110 (0.095)	-2.619 (0.266)	1.240 (0.591)	-0.641 (0.247)	-2.590 (0.271)
	$\rho = 10.0$	0.917 (0.383)	-0.037 (0.041)	-2.117 (0.184)	0.979 (0.383)	-0.384 (0.139)	-2.137 (0.185)

Table 2: Table 1 continued.

$\alpha = 1$		$(+i)$			$(-i)$		
$\kappa = 0$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	3.025 (0.793)	-0.568 (0.189)	-4.505 (0.405)	3.040 (0.805)	-0.565 (0.186)	-4.518 (0.405)
	$\rho = 0.15$	3.900 (0.916)	-0.752 (0.338)	-4.753 (0.294)	3.930 (0.960)	-0.752 (0.335)	-4.758 (0.293)
	$\rho = 10.0$	3.760 (0.880)	-0.889 (0.437)	-4.649 (0.287)	3.738 (0.875)	-0.890 (0.431)	-4.648 (0.289)
$c = (2, 3)$	$\rho = 0.05$	2.976 (1.386)	-0.305 (0.267)	-3.100 (0.582)	3.014 (1.411)	-0.296 (0.267)	-3.101 (0.587)
	$\rho = 0.15$	3.029 (1.333)	-0.339 (0.264)	-3.182 (0.531)	3.082 (1.329)	-0.334 (0.265)	-3.190 (0.534)
	$\rho = 10.0$	2.819 (1.254)	-0.310 (0.225)	-3.081 (0.483)	2.823 (1.296)	-0.311 (0.223)	-3.078 (0.495)
$c = (4, 3)$	$\rho = 0.05$	1.856 (0.972)	-0.166 (0.171)	-2.666 (0.450)	1.925 (0.991)	-0.160 (0.166)	-2.696 (0.442)
	$\rho = 0.15$	1.833 (0.810)	-0.127 (0.135)	-2.523 (0.400)	1.863 (0.804)	-0.124 (0.137)	-2.537 (0.404)
	$\rho = 10.0$	1.532 (0.718)	-0.065 (0.090)	-2.191 (0.335)	1.546 (0.719)	-0.064 (0.087)	-2.194 (0.337)
$\kappa = 0.5$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	3.059 (1.243)	-0.830 (0.242)	-4.888 (0.487)	3.326 (1.104)	-0.828 (0.237)	-4.897 (0.447)
	$\rho = 0.15$	3.622 (1.236)	-0.827 (0.324)	-5.035 (0.409)	3.948 (1.025)	-0.882 (0.318)	-5.045 (0.394)
	$\rho = 10.0$	3.628 (1.211)	-0.981 (0.449)	-4.858 (0.359)	4.009 (1.032)	-0.998 (0.437)	-4.930 (0.335)
$c = (2, 3)$	$\rho = 0.05$	2.854 (1.293)	-0.246 (0.272)	-3.317 (0.556)	3.039 (1.281)	-0.240 (0.262)	-3.356 (0.533)
	$\rho = 0.15$	2.846 (1.256)	-0.313 (0.280)	-3.402 (0.528)	3.051 (1.309)	-0.312 (0.273)	-3.428 (0.538)
	$\rho = 10.0$	2.674 (1.270)	-0.313 (0.248)	-3.293 (0.563)	2.859 (1.193)	-0.320 (0.242)	-3.320 (0.547)
$c = (4, 3)$	$\rho = 0.05$	1.913 (0.916)	-0.150 (0.168)	-3.073 (0.419)	1.994 (0.920)	-0.134 (0.154)	-3.089 (0.423)
	$\rho = 0.15$	1.796 (0.791)	-0.167 (0.152)	-2.984 (0.405)	1.883 (0.791)	-0.153 (0.139)	-2.997 (0.400)
	$\rho = 10.0$	1.633 (0.666)	-0.152 (0.116)	-2.727 (0.321)	1.710 (0.684)	-0.154 (0.122)	-2.737 (0.332)
$\kappa = 1$		LT	IT	LP	LT	IT	LP
$c = (1, 1)$	$\rho = 0.05$	2.838 (0.839)	-0.595 (0.188)	-4.507 (0.408)	2.684 (0.811)	-1.071 (0.244)	-4.363 (0.411)
	$\rho = 0.15$	3.642 (0.908)	-0.761 (0.339)	-4.761 (0.294)	3.353 (1.004)	-1.291 (0.408)	-4.563 (0.327)
	$\rho = 10.0$	3.759 (0.864)	-0.910 (0.436)	-4.659 (0.284)	2.785 (1.113)	-1.278 (0.461)	-4.403 (0.325)
$c = (2, 3)$	$\rho = 0.05$	2.765 (1.437)	-0.334 (0.268)	-3.094 (0.562)	2.167 (1.355)	-1.146 (0.520)	-2.977 (0.555)
	$\rho = 0.15$	2.798 (1.360)	-0.365 (0.280)	-3.163 (0.550)	2.288 (1.245)	-1.280 (0.499)	-3.071 (0.469)
	$\rho = 10.0$	2.575 (1.363)	-0.344 (0.235)	-3.054 (0.503)	2.201 (1.244)	-1.288 (0.404)	-3.000 (0.451)
$c = (4, 3)$	$\rho = 0.05$	1.770 (0.977)	-0.208 (0.198)	-2.673 (0.432)	1.239 (0.940)	-0.785 (0.377)	-2.588 (0.441)
	$\rho = 0.15$	1.745 (0.866)	-0.148 (0.156)	-2.526 (0.400)	1.353 (0.843)	-0.749 (0.340)	-2.480 (0.387)
	$\rho = 10.0$	1.324 (0.766)	-0.071 (0.097)	-2.188 (0.341)	1.362 (0.774)	-0.647 (0.259)	-2.209 (0.351)

Table 3: Table 1 continued.

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: either when an input is removed (-i), or when a useless input is added (+i), 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 read 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 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 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. We suppose that it learns information about the structure of the data it is later fed from. Considering that, in this case, A_{f_2} and A_d both start with similar performance, but A_{f_2} learns more efficiently than A_d, we suggest that this phenomenon be studied as an alternative parameter initialization procedure. A parsimonious hypothesis is that A_{f_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 PL 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 PL would be insensitive to it; or that the influence of ρ on PL is too complex to be figured out with the results of this simple experiment only.

According to TL theory [42] (Section 1.2), 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 (32) is an aggregated estimation of these three possible advantages, and IT (31) 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 PL in this experiment. This was an open question in [3], and the effect C is another argument in this favor. In the future, we expect not only *Negative Transfer* to be addressed with these measures, but also *Catastrophic Forgetting* with conservative approaches of PL, and experimental runs reverting and (+i) and (-i) events with subsequent (-i) and (+i) events during learning.

The protean approach was compared to a naive baseline resuming the learning from scratch on the change event. There are other unadressed obvious baselines, like the strategy simply ignoring the additional input forever after (+i), whose asymptotic performances are expected to be weaker when a new critical information is gained, but similar to the protean approach if the additional input is useless. In the future, we expect more refined protean strategies to be compared to less naive baselines. For instance, the negative transfer

perturbation observed in E is possibly mitigated by a mixed strategy only giving full control to the new learning model once the peak has decayed.

Only one change event (+i) or (-i) was tested on every run, so it remains unclear how learning reacts to successive events, or "multiple" events where *several* inputs are added or removed at once. Considering the results above, we expect that the reaction to successive changes be magnified by their frequency relative to the duration of the transient perturbation when uninformative inputs are added or removed. When adding or removing a bunch of inputs at once, we expect that the reaction ultimately depends on the usefulness of the information brought by the bunch compared to the one available in the stable inputs, although further experimentation or theoretical groundwork is needed to precisely quantify this effect.

From a more general perspective, 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. For these preliminary results to be extended to full-fledged RL, they still need to confront to the challenges of incremental learning, where the data appears piece by piece and not in full batch samples from a static training set [28]. In other words, the agent learning and optimization timeline (horizontal axis Fig. 4) and the input data stream timeline (horizontal axis Fig. 3) are not the same in OSL, but they need to line up in RL. With full-fledged RL, the credit assignment problem [40] also has to be met. In [2], we extend and complete the above results by addressing the protean reaction to (+i) and (-i) in a tabular RL benchmark, featuring 8 different protean profiles instead of 3. The same *natural projections* are used as a transfer technique in two traditional learning algorithms: Q-Learning and Actor-Critic. These extended results show the same significant advantage of the protean approach in a variety of learning situations. They also precisely outline the interplay between transfer and exploration in the case of PL, and suggest that future improvements of PL will follow from the dedication of transfer techniques to specific learning contexts. In other terms, we expect that the basic transfer techniques addressed in this work be useful as a generic PL "jacknife" when encountering (+i) and (-i) events, but are only the baseline of future improvements of PL.

Conclusion

RL agents are expected to continuously adapt to their environments, but real-world situations challenge them with changes in their input/output stream signature. We observe that the corresponding learning situation is not commonly tackled, or commonly acknowledged by the ML community. In this paper, we have generalized the idea of RL to Protean Learning (PL), a broader learning situation that explicitly takes the signature changes into account. We expect that PL be used as a starting point for research in this new area.

To defend PL, we have first presented a novel formal vision of RL, making it possible to extend to PL. This diagram-based formalism explicitly focuses on the streaming nature of learning, and especially highlights the transience of involved data sources, should they be input feed for the agent ((+i), (-i), ($\sim i$)), produced by the agent itself ((+o), (-o), ($\sim o$)) or by the users ((+ϕ), (-ϕ), ($\sim\phi$)). With this representation, the PL problem can be rigorously stated, well-defined and reasoned upon.

Then, with a controlled experiment, we have started exploring the development of PL learners. With traditional RNNs, and a low-level transfer technique borrowed from TL, we have built a new learner agent with a flexible input signature, aware of input transience, and we have shown that *input addition* and *input deletion* ((+i), (-i)) can be gracefully handled while learning non-Markovian and non-linear tasks, at least in an idealized OSL situation.

We have shown that this learner can reuse past experience to outperform a naive agent that restarts the learning from scratch on a signature change. And, with a statistical analysis of the learning traces, we have determined various detailed effects in play during the process, and how they relate to the properties of the transient input data.

These preliminary results are encouraging, as they suggest that agents can learn even if their agent-environment interface is changing. However, they only scratch the surface of a broad class of challenges raised by PL. First, we still need to verify that the results can be transposed outside the restricted context of OSL, so that agents could benefit from them in all RL situations. This is the object of current work in progress. Then, we also need to address the different types of signature changes, like *output addition* ($+o$), *feedback change* ($\sim\phi$), *merging* agents together *etc.* As the challenge associated with each new type of change differs much from the ones we have been concerned with ($+i$) and ($-i$) throughout this work, we think that they will require to be tackled with various different approaches. In addition, we have only tested one methodological approach of PL, based on RNNs and a low-level parameters transfer method, so we cannot guarantee that they yield the best results against PL, and other learning techniques still have to be compared. Finally, once the traditional RL benchmarks will have been extended to support signature changes and varying agent-environment interfaces, it will be a remaining challenge to make protean learners sufficiently flexible to adapt the need to deal with changing interfaces in real-world applications.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Abramowitz, M., Stegun, I.A.: Lengendre Functions. In: Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables, 9th edn., pp. 771–802. New York: Dover (1972)
2. Bonnici, I.: Towards Protean Learning: Accommodating Signature Changes in Artificial Agents. PhD Thesis, Université de Montpellier (2021)
3. Bonnici, I., Gouaïch, A., Michel, F.: Effects of Input Addition in Learning for Adaptive Games: Towards Learning with Structural Changes. In: EvoApplications: Applications of Evolutionary Computation, vol. LNCS, pp. 172–184. Leipzig, Germany (2019). DOI 10.1007/978-3-030-16692-2_12
4. Busto, P.P., Gall, J.: Open Set Domain Adaptation. In: International Conference on Computer Vision (ICCV), pp. 754–763. IEEE, Venice (2017). DOI 10.1109/ICCV.2017.88
5. Caruana, R.: Learning Many Related Tasks at the Same Time with Backpropagation. In: 7th International Conference on Neural Information Processing Systems, NIPS'94, pp. 657–664. MIT Press, Denver, Colorado (1994)
6. Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bengio, Y., Bahdanau, D.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar (2014). DOI 10.3115/v1/D14-1179
7. Cui, Y., Ahmad, S., Hawkins, J.: Continuous Online Sequence Learning with an Unsupervised Neural Network Model. Neural Computation **28**(11), 2474–2504 (2016). DOI 10.1162/NECO_a_00893
8. De Rosario-Martinez, H.: Phia: Post-Hoc Interaction Analysis (2015)
9. Devin, C., Gupta, A., Darrell, T., Abbeel, P., Levine, S.: Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer. In: International Conference on Robotics and Automation (ICRA), pp. 2169–2176. IEEE, Singapore, Singapore (2017). DOI 10.1109/ICRA.2017.7989250
10. Elman, J.L.: Finding structure in time. Cognitive Science **14**(2), 179–211 (1990). DOI 10.1016/0364-0213(90)90002-E
11. Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, J.: Meta Learning Shared Hierarchies. ArXiv **abs/1710.09767** (2018)

12. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Computing Surveys **46**(4), 1–37 (2014). DOI 10.1145/2523813
13. Ge, L., Gao, J., Ngo, H., Li, K., Zhang, A.: On handling negative transfer and imbalanced distributions in multiple source transfer learning: Multiple Source Transfer Learning. Statistical Analysis and Data Mining: The ASA Data Science Journal **7**(4), 254–271 (2014). DOI 10.1002/sam.11217
14. Gu, S., Lillicrap, T., Sutskever, I., Levine, S.: Continuous Deep Q-Learning with Model-based Acceleration. In: M.F. Balcan, K.Q. Weinberger (eds.) 33rd International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 48, pp. 2829–2838. PMLR, New York, New York, USA (2016)
15. Gupta, A.K., Smith, K.G., Shalley, C.E.: The Interplay Between Exploration and Exploitation. Academy of Management Journal **49**(4), 693–706 (2006). DOI 10.5465/amj.2006.22083026
16. Hanna, C.J., Hickey, R.J., Charles, D.K., Black, M.M.: Modular Reinforcement Learning architectures for artificially intelligent agents in complex game environments. In: Symposium on Computational Intelligence and Games (CIG), pp. 380–387. IEEE, Copenhagen, Denmark (2010). DOI 10.1109/ITW.2010.5593329
17. Harel, M., Mannor, S.: Learning from Multiple Outlooks. In: 28th International Conference on International Conference on Machine Learning, ICML'11, pp. 401–408. Omnipress, Madison, WI, USA (2011)
18. Heng Wang, Abraham, Z.: Concept drift detection for streaming data. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–9. IEEE, Killarney, Ireland (2015). DOI 10.1109/IJCNN.2015.7280398
19. Hinton, G.E., Sejnowski, T.J. (eds.): Unsupervised Learning: Foundations of Neural Computation. Computational Neuroscience. MIT Press, Cambridge, Mass (1999)
20. Jones, M.C., Marron, J.S., Sheather, S.J.: A Brief Survey of Bandwidth Selection for Density Estimation. Journal of the American Statistical Association **91**(433), 401–407 (1996). DOI 10.1080/01621459.1996.10476701
21. Kaplanis, C., Shanahan, M., Clopath, C.: Continual Reinforcement Learning with Complex Synapses. In: J. Dy, A. Krause (eds.) 35th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 80, pp. 2497–2506. PMLR, Stockholmsmässan, Stockholm Sweden (2018)
22. Khanıev, T.A., Unver, İ., Maden, S.: On the semi-Markovian random walk with two reflecting barriers. Stochastic Analysis and Applications **19**(5), 799–819 (2001). DOI 10.1081/SAP-120000222
23. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: Y. Bengio, Y. LeCun (eds.) 3rd International Conference on Learning Representations (ICLR). San Diego, CA, USA (2015)
24. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences **114**(13), 3521–3526 (2017). DOI 10.1073/pnas.1611835114
25. Lazaric, A.: Transfer in Reinforcement Learning: A Framework and a Survey. In: M. Wiering, M. van Otterlo (eds.) Reinforcement Learning, vol. 12, pp. 143–173. Springer (2012)
26. Li, W., Duan, L., Xu, D., Tsang, I.W.: Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(6), 1134–1148 (2014)
27. Lillicrap, T.P., Hunt, J.J., er Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv e-prints p. arXiv:1509.02971 (2015)
28. Losing, V., Hammer, B., Wersing, H.: Incremental on-line learning: A review and comparison of state of the art algorithms. Neurocomputing **275**, 1261–1274 (2018). DOI 10.1016/j.neucom.2017.06.084
29. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). DOI 10.1038/nature14236
30. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA (2012)
31. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch (2017)
32. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2020)
33. Ring, M.B.: Continual Learning in Reinforcement Environments. PhD Thesis, University of Texas at Austin, Austin, TX, USA (1994)
34. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive Neural Networks. CoRR **abs/1606.04671** (2016)

35. Saad, D. (ed.): On-Line Learning in Neural Networks. Publications of the Newton Institute. Cambridge University Press, Cambridge, [Eng.] ; New York (1998)
36. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015). DOI 10.1016/j.neunet.2014.09.003
37. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–503 (2016)
38. Sodhani, S., Chandar, S., Bengio, Y.: Toward Training Recurrent Neural Networks for Lifelong Learning. Neural Computation **32**(1), 1–35 (2020). DOI 10.1162/neco_a_01246
39. Spooner, T., Fearnley, J., Savani, R., Koukorinis, A.: Market Making via Reinforcement Learning. In: 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '18, pp. 434–442. International Foundation for Autonomous Agents and Multiagent Systems, Stockholm, Sweden (2018)
40. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, Mass (2018)
41. Tanaka, F., Yamamura, M.: 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, pp. 1108–1113. IEEE, Kobe, Japan (2003). DOI 10.1109/CIRA.2003.1222152
42. Taylor, M.E., Stone, P.: Transfer Learning for Reinforcement Learning Domains: A Survey. Journal of Machine Learning Research **10**(7), 1633–1685 (2009)
43. Teh, Y., Bapst, V., Czarnecki, W.M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., Pascanu, R.: Distral: Robust multitask reinforcement learning. In: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (eds.) Advances in Neural Information Processing Systems 30, pp. 4496–4506. Curran Associates, Inc. (2017)
44. Thrun, S.: Is Learning the N-th Thing Any Easier Than Learning the First? In: 8th International Conference on Neural Information Processing Systems, NIPS'95, pp. 640–646. MIT Press, Denver, Colorado (1995)
45. Torrey, L., Shavlik, J.: Transfer Learning. In: E.S. Olivas, J.D.M. Guerrero, M. Martinez-Sober, J.R. Magdalena-Benedito, A.J. Serrano López (eds.) Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, pp. 242–264. IGI Global (2010). DOI 10.4018/978-1-60566-766-9.ch011
46. Tsymbal, A.: The Problem of Concept Drift: Definitions and Related Work (2004)
47. Watanabe, C., Hiramatsu, K., Kashino, K.: Modular representation of layered neural networks. Neural Networks **97**, 62–73 (2018). DOI 10.1016/j.neunet.2017.09.017
48. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1), 69–101 (1996). DOI 10.1007/BF00116900
49. Xu, J., Zhu, Z.: Reinforced continual learning. In: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (eds.) Advances in Neural Information Processing Systems 31, vol. 31, pp. 899–908. Curran Associates, Inc. (2018)
50. Yang, Q., Chen, Y., Xue, G.R., Dai, W., Yu, Y.: 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, *ACL '09*, vol. 1, pp. 1–9. Association for Computational Linguistics, Suntec, Singapore (2009)
51. Yaochu Jin, Sendhoff, B.: Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **38**(3), 397–415 (2008). DOI 10.1109/TSMCC.2008.919172
52. Zenke, F., Poole, B., Ganguli, S.: Continual Learning Through Synaptic Intelligence. In: D. Precup, Y.W. Teh (eds.) 34th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 70, pp. 3987–3995. PMLR, International Convention Centre, Sydney, Australia (2017)
53. Žliobaitė, I., Pechenizkiy, M., Gama, J.: An Overview of Concept Drift Applications. In: N. Japkowicz, J. Stefanowski (eds.) Big Data Analysis: New Algorithms for a New Society, vol. 16, pp. 91–114. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-26989-4_4