Dynamic Modeling in Inductive Inference

John Case and Timo Kötzing*

Department of Computer and Information Sciences,
University of Delaware, Newark, DE 19716-2586, USA
{case,koetzing}@cis.udel.edu

Abstract. Introduced is a new inductive inference paradigm, *Dynamic Modeling*. Within this learning paradigm, for *example*, function h *learns* function g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be *programs for* the *output sequence* of g.

Dynamic Modeling provides an idealization of, for example, a social interaction in which h seeks to discover program models of g's behavior it sees in interacting with g, and h *openly* discloses to g its sequence of candidate program models to see what g says back.

Sample results: every g can be so learned by some h; there are g that can only be learned by an h if g can also learn that h back; there are extremely secretive h which cannot be learned back by any g they learn, but which, nonetheless, succeed in learning infinitely many g; quadratictime learnablity is strictly more powerful than lintime learnablity.

This latter result, as well as others, follow immediately from general correspondence theorems obtained from a *unified* approach to the paradigms within inductive inference.

Many proofs, some sophisticated, employ machine self-reference, a.k.a., recursion theorems.

1 Introduction and Motivation

In **Computational Limit Learning of Computable Functions** mapping the non-negative integers to same, an *algorithmic learner* is iteratively given more and more *finite* information generated by a *computable target function*. From this information, the learner, in each iteration, (may) synthesize a (suitably interpreted) natural number as output. In the literature, many criteria of successful learning have been proposed. Each such learning criterion defines precisely, possibly among other things, in what way the information will be generated by the target function and how the sequence of iteratively generated outputs and the target function have to relate for the learning to be considered successful. Sometimes each output number will be interpreted as (numerically naming) a program, other times each

* The authors would like to thank Samuel E. Moelius III for various fruitful discussions, especially on reactive learnees.

Y. Freund et al. (Eds.): ALT 2008, LNAI 5254, pp. 404–418, 2008.

number will represent a prediction for a yet unseen data point. It is helpful for the present paper to briefly consider some illustrative examples.

Ex-learning [Gol67] exemplifies the category of *identification*. h Ex-learns target g iff, in the i-th iteration, h outputs a conjecture on input $g(0)\dots g(i-1)$ and there are j, e such that $\forall k \geq j : h(g(0)\dots g(k-1)) = e$ and e is a program for g.[1] Such a program e could be carried away and used *offline*.

Nv-learning [Bār71, BB75] exemplifies the category of *extrapolation*. h Nv-learns target g iff, h is total and, in the i-th iteration, h outputs a conjecture on input $g(0)\dots g(i-1)$ and there is a j such that $\forall k \geq j : h(g(0)\dots g(k-1)) = g(k)$.[2] The successful extrapolants $h(g(0)\dots g(k-1))$, $k \geq j$, can be used *online*.

Coord-learning [MO99] exemplifies the category of *coordination*. h Coord-learns a target g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be the same as the sequence of g's outputs. The finally successfully coordinated matching outputs can be used *online*.

We see above that, while Coord-learning features a *reactive learnee* g, Ex- and Nv-learning feature a *passive learnee* g.

	Passive Learnee	**Reactive Learnee**
Off-Line	Identification	??
On-Line	Extrapolation	Coordination

In the just above table, there is a missing, not heretofore studied category entry for *offline* with *reactive learnee*. We refer to this category as *dynamic modeling*, and it is the subject of the present paper. **XBc**-learning exemplifies the category of *dynamic modeling*. h **XBc**-learns a target g iff, in the i-th iteration, h and g both produce output, h gets the sequence of all outputs from g in prior iterations as input, g gets all the outputs from h in prior iterations as input, and, from some iteration on, the sequence of h's outputs will be *programs for* the *output sequence* of g.[3]

In cognitive science, *theory of mind* refers to ones having a model (or models) of another's thoughts, emotions, and perspectives — including those different from ones own. Ideally, one might have a *program* (or programs) generating the behavior of the other, but — the behavior presented by the other would, in reality, be all and only that resulting from crossfeeding between oneself and the other. While one is attempting to synthesize program(s) for the other, *a* technique to employ

[1] The term 'Ex' stands for *explanatory* [CS83].

[2] The term 'Nv' stands for *next value* [Bār71].

[3] **X** we pronounce *cross*, and it is short for *crossfeed*. Of course crossfeeding of data is common to both the categories of coordination and dynamic modeling. In Section 3 we use the **X** also in talking about the former category. **Bc** stands for *behaviorally correct* [CS83].

is to pass on a sequence of remarks such as, "I think you are like ...," (where ... might be a program), and, then, attend to the resultantly elicited sequence of reactions of the other — as one formulates further programs/models of the other. Of course, in reality, one might, in seeking social understanding, carry out variants, including highly filtered variants, of the just above scenario. The unfiltered and very idealized scenario above is, nonetheless, covered by dynamic modeling.

Next we summarize the contents of the remaining sections.

In Section 2 below we present mathematical preliminaries.

Section 3 presents a *unified* approach to limiting learning criteria. This pays off in Section 5 where we can *then* provide general results applying to many criteria at once and, thereby, *quickly* obtain some nice corollaries.[4]

Section 4 involves cooperativeness vs. secretiveness in dynamic modeling. Considered are dynamic modelers which may or may not, in return, be dynamically modeled themselves. Proposition 4 implies that *no* computable g can keep models of its behavior totally secret; moreover, for any computable g, there are infinitely many constant functions h that **XBc**-learn g.[5]

Surprisingly, Theorem 6 implies that there is a computable g so that, *no* computable h *that* **XBc**-*learns* g can keep models of its behavior a secret from g, i.e., such h gives itself away: g, in turn, **XBc**-learns h. Positively, such a g is, then, *extremely cooperative*: informally, g can figure out the behavior of any computable h that figures out its behavior. Furthermore, such a g can be chosen to be lintime computable! The proof (included) of Theorem 6 is particularly elegant.

We say that computable h is *extremely uncooperative* iff $\{\text{computable } g \mid h \textbf{ XBc}\text{-learns } g \ \wedge \ g \textbf{ XBc}\text{-learns } h\} = \emptyset$. Theorem 10 implies there are extremely uncooperative computable h which, nonetheless, are infinitely successful, i.e, such that h **XBc**-learns infinitely many computable g.

Results in Section 4 feature open disclosure of certain learners' models of another while not disclosing their own models to the other. For comparison and contrast, a *zero-knowledge proof* [BSMP91] permits open, convincing disclosure of its existence without disclosing how it works.

Section 5 features two general and powerful correspondence theorems (Theorems 17 and 19) regarding many of the criteria discussed above and in Section 3.

The first *immediately* yields Corollary 18 which implies, for *example*, that quadratictime **XBc**-learnablity is strictly more powerful than lintime **XBc**-learnablity.[6]

[4] In Section 3 Ex-learning will be called **GEx**-learning, where the **G** is for Gold [Gol67]. Nv-learning will be called $\mathcal{R}$**GM**, where the $\mathcal{R}$ is for (total) computable learner and learnee, and the **M** is for Matching. Coord-learning will be called **XM**-learning.

[5] Actually, Proposition 4 is stated for a more restrictive criterion within the dynamic modeling category.

[6] Nothing like this happens for, for example, Ex-learning, since by an extension of Pitt's postponement tricks from [Pit89], (otherwise unrestricted) Ex-learning with lintime learners is just as powerful as Ex-learning. As we'll see from Theorem 14, also in Section 5, such extended postponement tricks *do*, nonetheless, apply to (even a restricted special case of) the **XBc**-learning of *any computably enumerable set of computable functions*.

Theorem 19 *immediately* yields Corollary 20 which provides a number of learning criteria hierarchies and separations. An *example*: the powers of **XBc**-learning and of Coord-learning are incomparable.

Many proofs are left out because of space constraints, and many proofs, some sophisticated, employ machine self-reference techniques, including Kleene Recursion Theorem (**krt**) [Rog67, page 214, problem 11-4] and Case's Operator Recursion Theorem (**ort**) [Cas74, Cas94]. The latter achieves infinitary self (and other) reference. See www.cis.udel.edu/~case/papers/DynamicModelingTR.pdf for a more complete version of the present paper.

2 Mathematical Preliminaries

Any unexplained complexity-theoretic notions are from [RC94]. All unexplained general computability-theoretic notions are from [Rog67].

$\mathbb{N}$ denotes the set of natural numbers, {0,1,2,...}. $*$ is a symbol such that, for all $n \in \mathbb{N}$, $n < *$. For two functions f, g and $n \in \mathbb{N}$, $f =^n g$ means that f and g differ on at most n arguments, $f =^* g$ means that f and g differ only on finitely many arguments.

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets. $\mathfrak{P}$ and $\mathfrak{R}$, respectively, denote the set of all partial ans total functions $\mathbb{N} \to \mathbb{N}$, respectively.

The quantifier $\forall^\infty x$ means "for all but finitely many x", the quantifier $\exists^\infty x$ means "for infinitely many x".

We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n \,.\, f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x \,.\, c$ is the constantly c function of one argument.

A function ψ is *partial computable* iff there is a Turing machine computing ψ. $\mathcal{R}$ and $\mathcal{P}$ denote the set of all (total) computable and partial computable functions $\mathbb{N} \to \mathbb{N}$, respectively. If ψ is not defined for some argument x, then we denote this fact by $\psi(x)\uparrow$, and we say that ψ on x *diverges*. The opposite is denoted by $\psi(x)\downarrow$, and we say that ψ on x converges.

We say that a partial function ψ *converges to* p iff $\forall^\infty x : \psi(x)\downarrow = p$.

[RC94, §3] describes an *efficiently* numerically named or coded[7] programming system for multi-tape Turing machines (TMs) which compute the partial computable functions $\mathbb{N} \to \mathbb{N}$. Herein we name this system φ. φ_p denotes the partial computable function computed by the TM-program with code number p in the φ-system, and Φ_p denotes the partial computable *runtime* function of the TM-program with code number p in the φ-system. By [RC94, Lemma 3.13], s-m-n, which provides for algorithmic storage of data in programs, has a *lintime* instance in this system. Hence, it can be shown that this system also has a lintime instance of 1-1 **ort**.

[7] This numerical coding guarantees that many simple operations involving the coding run in linear time. This is by contrast with historically more typical codings featuring prime powers and corresponding at least exponential costs to do simple things.

Whenever we consider tuples of natural numbers as input to functions, it is understood that a fixed *efficient* pairing function $\langle \cdot, \cdot \rangle$ (as in [RC94]) is used to code (left-associatively) the tuples into a single natural number.

A *finite sequence* is a mapping with a finite initial segment of $\mathbb{N}$ as domain (and range, $\mathbb{N}$). $\emptyset$ denotes the empty sequence (and, also, the empty set). The set of all finite sequences is denoted by $\mathbb{S}\mathrm{eq}$. For each finite sequence σ, we will denote the first element, if any, of that sequence by $\sigma(0)$, the second, if any, by $\sigma(1)$ and so on. $\#\mathrm{elets}(\sigma)$ denotes the number of elements in a finite sequence σ, that is, the cardinality of its domain. $\mathrm{last}(\sigma)$ denotes the last element of σ (if any).

For a partial function g and $i \in \mathbb{N}$, if $\forall j < i : g(j)\downarrow$, then $g[i]$ is defined to be the finite sequence $g(0), \ldots, g(i-1)$.

We take $\mathbb{N} = \{0,1\}^*$ for ease of measurement of the size of each natural number. Following [LV97], we fix an easily computed and inverted coding of all finite sequences of natural numbers into $\mathbb{N}$ so that the size of any sequence, defined as the size of its coding, is sensible for measuring the computational complexity of functions which take sequences as inputs. In particular the size of any sequence σ should be linear in $\#\mathrm{elets}(\sigma)$ and the size of each natural number in σ.

We let **LinF**, **PF** and **EXPF** be the set of all lintime, polytime and exptime computable functions, respectively.

Henceforth, we will many times identify a finite sequence σ with its code number $\langle \sigma \rangle_{\mathbb{S}\mathrm{eq}}$. However, when we employ expressions such as $\sigma(x)$, $\sigma = f$ and $\sigma \subset f$, we consider σ as a *sequence*, not as a number.

There are 1-1 function pad and function unpad_1, both $\in$ **LinF** and such that for all n, e: $\varphi_{\mathrm{pad}(n,e)} = \varphi_e$ and $\mathrm{unpad}_1(\mathrm{pad}(n,e)) = n$.

3 Unified Approach to Limit Learning Criteria

In this section, in the interest of generality, we give many definitions for limit learning also involving *non*-algorithmic learning. Nonetheless, all of the *results* given in the present paper concern only *algorithmic* (including complexity bounded) learning.

3.1 Definitions

Any set $\mathcal{C} \subseteq \mathfrak{P}$ is a *learner admissibility restriction*; intuitively, a learner admissibility restriction defines which functions are admissible as potential learners, e.g., $\mathcal{P}, \mathcal{R}, \mathbf{LinF}, \ldots$.

Every computable operator $\mathfrak{P} \times \mathfrak{R} \to \mathfrak{P}^2$ is called a *sequence generating operator*; intuitively, a sequence generating operator defines how learner and learnee interact to generate two infinite sequences, one for learner-outputs (we call this sequence the *learner-sequence*) and one for learnee-outputs.[8] For example, for

[8] Essentially, *these* computable operators are the recursive operators of [Rog67] but with two arguments and two outputs and restricted to the indicated domain.

Ex-learning (to be renamed in Section 3.2), for a (then, passive) learnee g, its learnee outputs would be $g(0), g(1), \ldots$. Below, in general, even for reactive learnees, we refer to the sequence of learnee-outputs as a *data-sequence*.

For $h \in \mathcal{P}, g \in \mathcal{R}$ and β a sequence generating operator, we call the first component of $\beta(h,g)$ the *learner-sequence* of h given g (denoted by $\beta_1(h,g)$), the second the *data-sequence* of g given h (denoted by $\beta_2(h,g)$).

Every subset of $\mathfrak{P}^2$ is called a *sequence acceptance criterion*. Intuitively, a sequence acceptance criterion defines which learner-sequences are considered a successful learning of a data-sequence. Any two such sequence acceptance criteria δ and δ' can be combined by intersecting them. For ease of notation we write $\delta\delta'$ instead of $\delta \cap \delta'$.

A *learning criterion* (or short *criterion*) is a 3-tuple consisting of a learner admissibility restriction, a sequence generating operator and a sequence acceptance criterion. Let $\mathcal{C}$, β, δ, respectively, be a learner admissibility restriction, a sequence generating operator and a sequence acceptance criterion, respectively. For $h \in \mathfrak{P}, g \in \mathfrak{R}$ we say that h $(\mathcal{C},\beta,\delta)$-learns g iff $h \in \mathcal{C}$ and $\beta(h,g) \in \delta$. For $h \in \mathfrak{P}$ and $\mathcal{S} \subseteq \mathfrak{R}$ we say that h $(\mathcal{C},\beta,\delta)$-learns $\mathcal{S}$ iff, for all $g \in \mathcal{S}$, h $(\mathcal{C},\beta,\delta)$-learns g. The set of $(\mathcal{C},\beta,\delta)$-learnable sets of computable functions is

$$\mathcal{C}\beta\delta := \{\mathcal{S} \subseteq \mathcal{R} \mid \exists h \in \mathfrak{P} : h \ (\mathcal{C},\beta,\delta)\text{-learns } \mathcal{S}\}. \tag{1}$$

We refer to the sets $\mathcal{C}\beta\delta$ as in (1) as *learnability classes*. Instead of writing the tuple $(\mathcal{C},\beta,\delta)$, we will ambiguously write $\mathcal{C}\beta\delta$. For $h \in \mathfrak{P}$, the set of all computable learnees $(\mathcal{C},\beta,\delta)$-learned by h is denoted by $\mathcal{C}\beta\delta(h) := \{g \in \mathcal{R} \mid h \ (\mathcal{C},\beta,\delta)\text{-learns } g\}$.

For any sequence generating operator β, we can turn a given sequence acceptance criterion δ into a learner admissibility restriction $\mathcal{T}_\beta\delta$ by admitting only those learners that obey δ *on all possible input*:

$$\mathcal{T}_\beta\delta := \{h \in \mathfrak{P} \mid \forall g \in \mathfrak{R} : \beta(h,g) \in \delta\}.$$

3.2 Examples

In this section we give many examples illustrating our definitions and give an overview as to how our notation covers criteria from the literature. Past this section, we will not be concerned with *every* example given in this section, but some of them will be employed.

Example 1. *Two typical learner admissibility restrictions are $\mathcal{P}$ and $\mathcal{R}$. Furthermore, any set of functions computable with a resource restriction (such as the set of all lintime computable functions) may be used as a learner admissibility restriction. For each sequence generating operator β and each $\mathfrak{F} \subseteq \mathfrak{R}$, the set* $\mathrm{Rel}_{\beta,\mathfrak{F}}$ *of all functions reliable on $\mathfrak{F}$ [Min76, BB75] (defined below) is also a learner admissibility restriction.*

$$\mathrm{Rel}_{\beta,\mathfrak{F}} := \{h \in \mathcal{P} \mid \forall g \in \mathfrak{F}\, \forall p,q \in \mathcal{P} : (\beta(h,g) = (p,q) \\ \wedge\ p \text{ converges to some } p' \ ) \Rightarrow \varphi_{p'} = q\}.$$

When denoting criteria with $\mathcal{P}$ as the learner admissibility restriction, we will omit $\mathcal{P}$.

Example 2. *We define the following example sequence generating operators. All learners give an initial conjecture, say, of 0, based on no data.*

- *Goldstyle [Gol67]:* $\mathbf{G} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (\lambda i . h(g[i]), g)$.
- *Iterative [Wie76]:* $\mathbf{It} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (p, q)$ *such that* $p(0) = 0$, $\forall n : p(n+1) = h(q(n), p(n))$ *and* $q = g$.
- *Transductive:* $\mathbf{Td} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{R}, (h, g) \mapsto (p, q)$ *such that* $p(0) = 0$, $\forall n : p(n+1) = h(q(n))$ *and* $q = g$.
- *Crossfeeding [MO99]* $\mathbf{X} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{P}, (h, g) \mapsto (p, q)$ *such that* $\forall n\ p(n) = h(q[n]) \wedge q(n) = g(p[n])$.
- *Learnee Iterative:* $\mathbf{Li} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{P} \times \mathfrak{P}, (h, g) \mapsto (p, q)$ *such that* $\forall n\ p(n) = h(q[n]) \ \wedge q(0) = 0 \ \wedge \forall n : q(n+1) = g(p(n), q(n))$.

"**G**" is a reference to Gold [Gol67]. Intuitively, **G** takes a learner h and a learnee g, and feeds longer and longer initial segments of g into h, considering the successive outputs as coding an infinite sequence of hypotheses. The second output is just g, meaning that the target concept to be learned is all of g. In this setting, the learner gets a lot of information about the learnee, while the learnee does not react at all to the learning process. For **It** and **Td** defined above, a learner for the latter has less information at its disposal than for the former.

Regarding **X**, learner and learnee have symmetrical information in each iteration. **Li** lessens the information that the learnee has in a similar way that iterative learning lessens the information of the learner.

The first three bullets given in Example 2 involve passive learnees, while the last two examples involve reactive learnees.

We note the following three important properties relating **G** and **X**, which are of importance to this paper. Let $f, g, h, p, q \in \mathcal{P}$.

$$\mathbf{X}(h, g) = (p, q) \Rightarrow \mathbf{G}(h, q) = (p, q). \tag{2}$$

$$\mathbf{X}(h, g) = (p, q) \Rightarrow \mathbf{G}(g, p) = (q, p). \tag{3}$$

$$\mathbf{X}(h, \lambda\sigma . f(\#\text{elets}(\sigma))) = \mathbf{G}(h, f). \tag{4}$$

Example 3. *We define the following sequence acceptance criteria.*

- *Explanatory:* $\mathbf{Ex} = \{(p, q) \in \mathfrak{P}^2 \mid p \text{ converges to some } p' \text{ and } \varphi_{p'} = q\}$.
- *Explanatory with up to* $a \in \mathbb{N} \cup \{*\}$ *errors [CS83, BB75]:* $\mathbf{Ex}^a = \{(p, q) \in \mathfrak{P}^2 \mid p \text{ converges to some } p' \text{ and } \varphi_{p'} =^a q\}$.
- *Behaviorally correct [CS83, Bār74b]:* $\mathbf{Bc} = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n\ \varphi_{p(n)} = q\}$.
- *Behaviorally correct with up to* $a \in \mathbb{N} \cup \{*\}$ *errors [CS83]:* $\mathbf{Bc}^a = \{(p, q) \in \mathfrak{P}^2 \mid \forall^\infty n\ \varphi_{p(n)} =^a q\}$.
- *Matching [Bār71, BB75, MO99]:* $\mathbf{M} = \{(p, q) \in \mathfrak{P} \times \mathfrak{R} \mid p =^* q\}$.

All the above criteria include global restrictions on the path to successful learning. The following defines several criteria only involving local restrictions.

- *Postdictively complete [Bār74a, BB75, Wie76]:* $\mathbf{Pcp} = \{(p,q) \in \mathfrak{R}^2 \mid \forall n \forall i < n : \varphi_{p(n)}(i) = q(i)\}$.
- *Hypotheses are programs for total functions [CS83]:* $\mathbf{T} = \{(p,q) \in \mathfrak{R}^2 \mid \forall n : \varphi_{p(n)} \in \mathcal{R}\}$.
- *Always giving hypotheses:* $\mathfrak{R}^2$.

The idea of dividing a learning criterion is not entirely new. For example, Freivalds et. al. [FKS95] defined *admissible sequences for a given function*, which basically defines a binary predicate on a pair of infinite sequences.

We can now express several learning criteria as defined in the prior literature (left-hand-side below) with our notation system (right-hand-side below). Recall that the default learner admissibility restriction is $\mathcal{P}$; hence, all learning criteria displayed just below are for *algorithmic* learners.

$$\begin{aligned}
\text{Ex} &\leftrightarrow \mathbf{GEx}\\
\text{Bc} &\leftrightarrow \mathbf{GBc}\\
\text{Nv} &\leftrightarrow \mathcal{R}\mathbf{GM}\\
\text{Nv}' &\leftrightarrow \mathbf{G}\mathcal{R}^2\mathbf{M}\\
\text{Nv}'' \text{ ([Pod74])} &\leftrightarrow \mathbf{GM}\\
\text{Cons} &\leftrightarrow \mathbf{GPcpEx}\\
\text{R-Cons} &\leftrightarrow \mathcal{R}\mathbf{GPcpEx}\\
\text{T-Cons} &\leftrightarrow \mathcal{T}_{\mathbf{G}}\mathbf{PcpGEx}\\
\text{Reliable on } \mathcal{R} &\leftrightarrow \text{Rel}_{\mathcal{R}}\mathbf{GEx}\\
\text{It} &\leftrightarrow \mathbf{ItEx}\\
\text{learneable by a player ([MO99])} &\leftrightarrow \mathbf{XM}\\
\text{learneable by a total player ([MO99])} &\leftrightarrow \mathcal{R}\mathbf{XM}
\end{aligned}$$

A sequence acceptance criterion δ is said to be *degenerate* iff $\exists (p,q) \in \delta : p =^* \lambda x\,.\uparrow$. All sequence acceptance criteria given above are non-degenerate, and the authors don't know of any degenerate sequence acceptance criteria implicit in the prior literature. We conjecture that any degenerate sequence acceptance criteria would be useless to model learning. Hence, the present paper will solely focus on non-degenerate such criteria.

It is easy to see that, for non-degenerate δ, we have for all $\mathcal{C} \subseteq \mathcal{P}$ that the learnability classes $\mathcal{C}\mathbf{X}\delta$ and $\mathcal{C}\mathbf{X}\mathcal{R}^2\delta$ are *equal*.

Similarities between extrapolation (like **GM**) and coordination (like **XM**) have been pointed out in [CJM$^+$05]. In particular, *blind* learnees are defined as functions where each output only depends on the *length* of it's input, and with each function $g \in \mathcal{R}$, a blind learnee $g' = \lambda\sigma\,.g(\#\text{elets}(\sigma))$ is associated. The mapping $\Theta = \lambda g\,.g'$ is then a natural embedding of learnees in the **G**-sense to learnees in the **X**-sense, more formally, for all $\delta \subseteq \mathcal{P} \times \mathcal{R}$ and $\mathcal{S} \subseteq \mathcal{R}$,

$$\mathcal{S} \in \mathbf{G}\delta \Leftrightarrow \Theta(\mathcal{S}) \in \mathbf{X}\delta. \tag{5}$$

The special case of (5) with $\delta = \mathbf{M}$ is used in [CJM$^+$05].

4 Cooperation and Secretiveness

The main emphasis of the present paper, as seen in Section 1, features **XBc**-learning, but, based on the thinking of Section 3, one might wonder why we didn't

talk about **XEx**-learning. We'll talk about it now. Suppose h **XEx**-learns g. The learner-sequence of h interacting with g, is, then, a total, almost everywhere constant function. Suitable g, then, easily **XEx**-learn h.[9]

The proposition just below implies that, *any* computable g gets **XBc**-learned by some computable h. Hence, any g can have its secrets learned by some learner. The interesting thing, then, is whether, when h **XBc**-learns g, h can keep models of itself secret from g. This is considered in Theorem 6 further below.

Proposition 4. Let $g \in \mathcal{R}$. Then there are infinitely many (total) constant functions h **XBc**- (in fact, **XEx**-) learning g.

Proof. Let $n \in \mathbb{N}$. There is, by **krt**, e_n such that, with

$$p = \lambda x . \operatorname{pad}(n, e_n), \tag{7}$$

$$\forall x : \varphi_{e_n}(x) = g(p[x]). \tag{8}$$

Let $h_n \in \mathcal{R}$ such that

$$\forall \sigma : h_n(\sigma) = \operatorname{pad}(n, e_n). \tag{9}$$

There is $q \in \mathcal{R}$ such that $\mathbf{X}(h_n, g) = (p, q)$. Then we have, for all t and x, we have

$$\varphi_{p(t)}(x) \underset{(7)}{=} \varphi_{\operatorname{pad}(n,e_n)}(x) = \varphi_{e_n}(x) \underset{(8)}{=} g(p[x]) \underset{\text{choice of } q}{=} q(x). \tag{10}$$

Hence, h_n **XBc**-learns g. Trivially, we have for all $l \neq m$, $h_l \neq h_m$. This shows that there are infinitely many different (constant) functions **XEx**-learning g. □

Definition 5. We define the following sequence acceptance criteria.

- Cooperative Bc: $\mathbf{Bcc} = \{(p, q) \in \mathcal{R}^2 \mid \forall^\infty n \ \varphi_{p(n)} = q \ \wedge \ \forall^\infty n \ \varphi_{q(n)} = p\}$ $(= \mathbf{BcBc}^{-1})$.
- Secretive Bc: $\mathbf{Bcs} = \{(p, q) \in \mathcal{R}^2 \mid \forall^\infty n \ \varphi_{p(n)} = q \ \wedge \ \neg\forall^\infty n \ \varphi_{q(n)} = p\}$ $(= \mathbf{Bc}\overline{\mathbf{Bc}^{-1}})$.

Clearly, for all $h, g \in \mathcal{R}$, h **XBcc**-learns g iff, h **XBc**-learns g and g **XBc**-learns h; similarly, h **XBcs**-learns g iff, h **XBc**-learns g and g does *not* **XBc**-learn h.

It is easy to see that there are computable functions which are not **XBcc**-learnable, for example $\lambda\sigma . \#\text{elets}(\sigma)$.[10] At first glance, it seems likely that all computable functions can be **XBcs**-learned, as, by Proposition 4 above, for

[9] We have, more generally, that

$$\exists g \in \mathcal{R} \ \forall h \in \mathcal{R} : h \ \mathbf{XEx}\text{-learns } g \Rightarrow g \ \mathbf{XEx}\text{-learns } h. \tag{6}$$

[10] For all $g \in \mathcal{R}$, and $p, q \in \mathcal{R}$ such that $\mathbf{X}(\lambda\sigma . \#\text{elets}(\sigma), g) = (p, q)$, we have that p is the identity on $\mathbb{N}$; hence, $\lambda\sigma . \#\text{elets}(\sigma)$ does not **XBc**-learn g.

any given function g, there are infinitely many functions h **XBc**-learning g. We were, then, surprised that not all computable functions can be **XBcs**-learned, as seen below in Theorem 6. Intuitively, this theorem means that there is a $g \in \mathcal{R}$ such that, for all $h \in \mathcal{P}$, if h **XBc**-learns g, then h has to give away enough information about itself so that g will be able to **XBc**-learn h. Even more surprisingly, such a g can be chosen to be lintime computable! On the other hand, Theorem 6 also has a positive interpretation: it *is* possible to find a function g that will **XBc**-learn every function h that **XBc**-learns g – in other words, there are *extremely cooperative functions* that will cooperate with *any* function **XBc**-learning them. We denote the set of extremely cooperative functions with $EC := \{h \in \mathcal{R} \mid \forall g \in \mathcal{R} : g \textbf{ XBc}\text{-learns } h \Rightarrow h \textbf{XBc}\text{-learns } g\}$.[11]

Theorem 6 (Secretiveness Fails)

$$\exists g \in \mathbf{LinF} : \{g\} \notin \mathbf{XBcs}.$$

Proof. By 1-1, lintime **ort** there is 1-1 $g \in \mathbf{LinF}$ such that

$$\forall \tau, x : \varphi_{g(\tau)}(x) = \text{last}(g^{-1}(\varphi_{\text{last}(\tau)}(x+1))).^{12} \tag{11}$$

Let $h \in \mathcal{P}$ be such that $g \in \mathbf{XBc}(h)$. Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = (p, q)$. Since $(p, q) \in \mathbf{Bc}$, there is n_0 such that

$$\forall n \geq n_0 : \varphi_{p(n)} = q. \tag{12}$$

Claim: $\forall^\infty n : \varphi_{q(n)} = p$.

Proof. We have

$$\forall n \geq n_0 + 1, x : \varphi_{p(n-1)}(x+1) \underset{(12)}{=} q(x+1) \underset{\text{choice of } q}{=} g(p[x+1]). \tag{13}$$

Hence, for all $n \geq n_0 + 1$ and all x,

$$\begin{aligned} \varphi_{q(n)}(x) &\underset{\text{choice of } q}{=} \varphi_{g(p[n])}(x) \underset{(11)}{=} \text{last}(g^{-1}(\varphi_{p(n-1)}(x+1))) \\ &\underset{(13)}{=} \text{last}(g^{-1}(g(p[x+1]))) = \text{last}(p[x+1]) = p(x). \end{aligned} \tag{14}$$

□ (FOR CLAIM)

Hence, by the claim, $g \in \mathbf{XBcc}(h)$; therefore, $\{g\} \notin \mathbf{XBcs}$.

□ (FOR THEOREM)

[MO99] examined uncooperativeness of coordinators. In particular, two sets of total computable functions are constructed such that any learner learning *all* of the functions from one of the sets cannot coordinate with *any* function

[11] It is easy to see that $EC = \{h \in \mathcal{R} \mid \mathbf{XBcs}^{-1}(h) = \emptyset\}$.

[12] Note that $\varphi_{g(\tau)}(x)$ might be undefined for various reasons, for example last is not total. Furthermore, note that accessing g^{-1} is a valid use of **ort**.

from the other set. Furthermore, [CJM+05] extended this result showing that for all $k \geq 2$, one can find k such sets of uncooperative learners. Below, we give an analog of this result for cooperation in the **XBcc**-sense, where we give an infinite family of uncooperative sets, so that any learner that can **XBc**-learn *any* of the functions in one of the sets cannot **XBc**-learn *any* of the functions of any other set.

Theorem 7 (Incompatible Mutual Cooperation Camps). There is a 1-1 $e \in \mathcal{R}$ such that for all m, n, $\varphi_{e(m,n)}$ total and, defining $\mathcal{S}_n := \{\varphi_{e(m,n)} \in \mathcal{R} \mid m \in \mathbb{N}\}$, for each n all members of $\mathcal{S}_n$ **XBcc**-learn each other, while each function $h \in \mathcal{P}$ **XBcc**-learns functions from at most one of the sets in $\{\mathcal{S}_n \mid n \in \mathbb{N}\}$.

The theorem just above can be shown by an application of the Generalized Delayed Recursion Theorem [Cas74, Theorem 23].

As a contrast to the extremely cooperative functions as defined above, we say that $h \in \mathcal{R}$ is *extremely uncooperative* iff $\mathbf{XBcc}(h) = \emptyset$ (i.e., h cooperates with no function). The set of all extremely uncooperative functions is denoted by EU. Trivially, $EU \neq \emptyset$, as EU contains each function h that doesn't **XBc**-learn any function. Interestingly, EU is rather big in the sense that the closure under **LinF** composition of EU *is equal to* $\mathcal{R}$.[13] However, many of the functions $h \in EU$ will not **XBc**-model anything. We define a (computable) operator below, turning a given learner h into an uncooperative learner h', which intuitively doesn't lose too much of the learning power of h.

Furthermore, Theorem 10 below states the existence of $h \in EU$ with $\mathbf{XBc}(h)$ infinite.

Definition 8. For all $h \in \mathcal{R}$ there is, by lintime **ort**, $h' \in \mathbf{LinF}$ such that

$$\forall \sigma, x : \varphi_{h'(\sigma)}(x) = \begin{cases} \varphi_{h(\sigma)}(x), & \text{if } \sigma = \emptyset \ \vee \ \exists s \geq \#\text{elets}(\sigma), t : \\ & \varphi_{\varphi_{h(\sigma)}(s)}(t)\!\downarrow \neq h'(\varphi_{h(\sigma)}[t])\!\downarrow; \\ \uparrow, & \text{otherwise.} \end{cases} \tag{15}$$

Intuitively, h' makes conjectures mostly behaviorally equivalent to those of h, but modified so that the conjectures are definitely wrong as soon as the input seems to learn the outputs of h'.

Define $\Psi = \lambda h \in \mathcal{R}.h'$. N.B. For each $h \in \mathcal{R}$, $\Psi(h) \in \mathbf{LinF}$.

Lemma 9. Let $h \in \mathcal{R}$. Then $\mathbf{XBcc}(\Psi(h)) = \emptyset$.

Theorem 10 (Extremely Uncooperative Infinitely Successful Learners). There are functions $h \in \mathcal{R}$ such that $\mathbf{XBcs}(\Psi(h))$ is infinite, but $\mathbf{XBcc}(\Psi(h)) = \emptyset$ (i.e., no function **XBc**-learned by $\Psi(h)$ can **XBc**-learn $\Psi(h)$).

The next theorem intuitively implies that requiring a learner to be extremely uncooperative will decrease its learning power with respect to plain uncooperative learning.

Corollary 11. $EU\mathbf{XBcs} \subset \mathcal{R}\mathbf{XBcs}$.[14]

[13] Let $f \in \mathcal{R}$, let p be such that $\varphi_p = \lambda x.\uparrow$. Then $f' = \lambda x.\text{pad}(f(x), p) \in EU$. Define $a = \lambda x.\text{unpad}_1(x) \in \mathbf{LinF}$. Then $a \circ f' = f$.

[14] Less surprisingly, one can also show $EC\mathbf{XBcc} \subset \mathcal{R}\mathbf{XBcc}$.

5 General Crossfeeding

Most lemmas, propositions and theorems of this section carry over with slightly modified hypotheses to the case of **Li** in the place of **X**.

Just below is a proposition with corollary regarding which sequence acceptance criteria allow dynamical modeling all of $\mathcal{R}$.

Proposition 12. Let δ be a sequence acceptance criterion, let $\mathcal{C} \subseteq \mathcal{P}$. Then we have

$$\mathcal{R} \in \mathcal{C}\mathbf{X}\delta \Leftrightarrow \mathcal{R} \in \mathcal{C}\mathbf{G}\delta.$$

We get the following corollary by a theorem of Harrington, cited in [CS83].

Corollary 13

$$\mathcal{R} \in \mathbf{XBc}^*.$$

Gold [Gol67] introduced learning by enumeration. Analogous to, but harder to prove than the case for **G**-style learning, we have, by the next theorem that any computably enumerable set of (total) computable functions are **XEx**-modelable,

Theorem 14 (Dynamic Modeling by Enumeration). Let $r \in \mathcal{R}$ be an enumeration of program numbers of (total) computable functions. We have

$$\{\varphi_{r(n)} \mid n \in \mathbb{N}\} \in \mathbf{XEx}.^{15}$$

Proof. By **ort**, there are programs e, e', as well as an infinite enumeration $s \in \mathcal{R}$ of programs such that, with $h = \varphi_e$ and $u = \varphi_{e'}$,[16]

$$\forall \sigma : u(\sigma) = \mu k \leq \#\mathrm{elets}(\sigma)\,.\,\sigma \subseteq \mathbf{X}_2(h, \varphi_{r(k)}); \tag{17}$$

$$\forall m : \varphi_{s(m)} = \mathbf{X}_2(h, \varphi_{r(m)}); \tag{18}$$

$$\forall \sigma : h(\sigma) = s(u(\sigma)). \tag{19}$$

By induction, we can show $h, u \in \mathcal{R}$. Let $n \in \mathbb{N}$. Let $g = \varphi_{r(n)}$. We show that h **XEx**-learns g. Let $p, q \in \mathcal{R}$ be such that $\mathbf{X}(h, g) = (p, q)$. Obviously, for all j, $u(q[j])\!\downarrow \leq n$. Also, u is monotone in the sense that $\forall i, j : i \leq j \Rightarrow u(q[i]) \leq u(q[j])$. Thus, there is m such that

$$\forall^\infty j : u(q[j]) = m. \tag{20}$$

[15] In fact, **Ex** could be replaced by any δ from a wide set of sequence acceptance criteria.

[16] For a number-theoretic (partial) predicate P and $n \in \mathbb{N}$, we let

$$\mu x \leq n\,.\,P(x) = \begin{cases} x, & \text{if } \forall y < x : P(y)\!\downarrow \neq \text{ true and } P(x)\!\downarrow = \text{ true;} \\ n+1, & \text{if } \forall y < n+1 : P(y)\!\downarrow \neq \text{ true;} \\ \uparrow, & \text{otherwise.} \end{cases} \tag{16}$$

Hence, p converges. By (17) and (20),

$$\forall^\infty j : q[j] \subseteq \mathbf{X}_2(h, \varphi_{r(m)}); \text{ thus,} \tag{21}$$
$$q = \mathbf{X}_2(h, \varphi_{r(m)}). \tag{22}$$

The following completes the proof.

$$\forall^\infty j : \ \varphi_{p(j)} \underset{\text{def } p}{=} \varphi_{h(q[j])} \underset{(19)\ \&\ (20)}{=} \varphi_{s(m)} \underset{(18)}{=} \mathbf{X}_2(h, \varphi_{r(m)}) \underset{(22)}{=} q. \tag{23}$$

□

The enumeration technique used in our proof of the theorem just above can be modified with techniques from [RC94] so as to obtain a lintime computable learner for any given computably enumerable set of functions. However, in general it is not the case that any **XEx**-learnable set is also **XEx**-learnable by a lintime learner. This will be stated formally in Corollary 18 below, for which we now give definitions to set it up.

Definition 15. Let δ be a sequence acceptance criterion.

- δ is called *non-trivial* iff $\forall \sigma : \sigma \diamond \mathcal{R} \notin \mathbf{G}\delta$.
- Let $\mathcal{D} \subseteq \mathcal{R}$. δ *allows for lintime path finding for* $\mathcal{D}$ iff there is $r \in \mathbf{LinF}$ such that, for all σ, τ of equal length and $q \in \mathcal{D}$ and e with $\sigma \subseteq q = \varphi_e$, we have $(\tau \diamond \lambda x . r(x, e, \sigma), q) \in \delta$

We illustrate the definitions by giving the following examples.

Example 16

- $\mathbf{Bc}^*$ *is a trivial sequence acceptance criterion, while* $\mathbf{Ex}, \mathbf{Bc}, \mathbf{Bcs}, \mathbf{Bcc}$ *and* $\mathbf{M}$ *are non-trivial.*
- $\mathbf{Ex}, \mathbf{Bc}$ *and* $\mathbf{Bc}^*$ *allow for lintime path finding for* $\mathcal{R}$ *as witnessed by* $r = \lambda x, e, \sigma . e$. $\mathbf{M}$ *allows for lintime pathfinding for total finite variants of constant functions.*[17]

Note that non-triviality is inherited by subsets, and allowing for lintime path finding by supersets.

Theorem 17 (Learner Correspondence). Let δ be non-trivial such that δ allows for lintime path finding for total finite variants of constant functions. Let $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{R}$ be closed under generalized composition with **LinF** and $\mathbf{LinF} \subseteq \mathcal{C}, \mathcal{C}'$. Then

$$\mathcal{C}\mathbf{X}\delta \subseteq \mathcal{C}'\mathbf{X}\delta \Leftrightarrow \mathcal{C} \subseteq \mathcal{C}'.$$

Suppose for discussion Q is a polynomial time bound. Pitt [Pit89] notes that polytime (update) Ex-learning allows unfair postponement tricks, i.e., a learner h can put off outputting significant conjectures based on data σ until it has

[17] $\mathbf{M}$ allows for not necessarily lintime path finding for $\mathcal{R}$.

seen a much larger sequence of data τ so that $Q(|\tau|)$ is enough time for h to think about σ as long as it needs. In fact, by an extension of Pitt's postponement tricks, $\mathbf{LinFGEx} = \mathbf{GEx}$. A direct application of Theorem 17, e.g., Corollary 18, implies that *dynamic modeling* does *not* allow for those kinds of postponement tricks *in general*.[18] We let α, as from [CLRS01, §21.4], be a *very slow growing*, unbounded, lintime computable function $\leq$ an inverse of Ackermann's function; let $\mathbf{LinF}^{+\varepsilon} := \{\varphi_e \in \mathcal{R} \mid \exists k \forall n : \Phi_e(n) \leq k \cdot |n| \cdot \log(|n|) \cdot \alpha(|n|) + k\}$. The classes $\mathbf{LinF}$ and $\mathbf{LinF}^{+\varepsilon}$ have long been known to separate [HS65].

The following corollary gives a sample of the universal power of Theorem 17.

Corollary 18 (Learner Complexity Matters). *Let* $\delta \in \{\mathbf{Ex}, \mathbf{Bc}, \mathbf{M}\}$.

(a) $\mathbf{LinFX}\delta \subset \mathbf{LinF}^{+\varepsilon}\mathbf{X}\delta$.
(b) $\mathbf{PFX}\delta \subset \mathbf{EXPFX}\delta$.

Theorem 17 can be generalized so as to show $\mathcal{R}\mathbf{X}\delta \subseteq \mathcal{P}\mathbf{X}\delta$ for δ as in Corollary 18.

Theorem 19 (Sequence Acceptance Correspondence). Let δ, δ' be such that $\delta \subseteq \mathcal{R}^2$ and δ' is non-trivial. We have

$$\mathbf{X}\delta \subseteq \mathbf{X}\delta' \Leftrightarrow \delta \subseteq \delta'.$$

The following corollary gives a sample of the universal power of Theorem 19.

Corollary 20 (Hierarchies and Separations)

(a) For all $a, b \in \mathbb{N} \cup \{*\} : \mathbf{XBc}^a \not\subseteq \mathbf{XEx}^b$.
(b) For all $a, b \in \mathbb{N} \cup \{*\} : \mathbf{XEx}^a \subseteq \mathbf{XBc}^b \Leftrightarrow a \leq b$.
(c) For all $n \in \mathbb{N} : \mathbf{XM} \not\subseteq \mathbf{XEx}^*, \mathbf{XBc}^n$.
(d) For all $n \in \mathbb{N} : \mathbf{XEx}^*, \mathbf{XBc}^n \not\subseteq \mathbf{XM}$.

References

[Bār71] Bārzdiņš, J.: Prognostication of automata and functions. Information Processing 1, 81–84 (1971)

[Bār74a] Bārzdiņš, J.: Inductive inference of automata, functions and programs. In: Int. Math. Congress, Vancouver, pp. 771–776 (1974)

[Bār74b] Bārzdiņš, J.: Two theorems on the limiting synthesis of functions. In: Theory of Algorithms and Programs, Latvian State University, Riga, vol. 210, pp. 82–88 (1974)

[18] However, as we saw from Theorem 14 above, such extended postponement tricks *do*, nonetheless, apply to the *special case of* the $\mathbf{XEx}$-learning of *any computably enumerable set of computable functions*.

However, we believe we can show that Theorem 14 doesn't hold for $\mathbf{LinFXPcpEx}$ in place of $\mathbf{LinFXEx}$; hence, postdictive completeness prevents some postponement tricks.

[BB75] Blum, L., Blum, M.: Toward a mathematical theory of inductive inference. Information and Control 28, 125–155 (1975)

[BSMP91] Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput. 20(6), 1084–1118 (1991)

[Cas74] Case, J.: Periodicity in generations of automata. Mathematical Systems Theory 8, 15–32 (1974)

[Cas94] Case, J.: Infinitary self-reference in learning theory. Journal of Experimental and Theoretical Artificial Intelligence 6, 3–16 (1994)

[CJM+05] Case, J., Jain, S., Montagna, F., Simi, G., Sorbi, A.: On learning to coordinate: Random bits help, insightful normal forms, and competency isomorphisms. Journal of Computer and System Sciences 71(3), 308–332 (2005); Special issue for selected learning theory papers from COLT 2003, FOCS 2003, and STOC 2003

[CS83] Case, J., Smith, C.: Comparison of identification criteria for machine inductive inference. Theoretical Computer Science 25, 193–220 (1983)

[CLRS01] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)

[FKS95] Freivalds, R., Kinber, E.B., Smith, C.H.: On the intrinsic complexity of learning. Information and Computation 123(1), 64–71 (1995)

[Gol67] Gold, E.: Language identification in the limit. Information and Control 10, 447–474 (1967)

[HS65] Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)

[LV97] Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)

[Min76] Minicozzi, E.: Some natural properties of strong identification in inductive inference. In: Theoretical Computer Science, pp. 345–360 (1976)

[MO99] Montagna, F., Osherson, D.: Learning to coordinate: A recursion theoretic perspective. Synthese 118, 363–382 (1999)

[Pit89] Pitt, L.: Inductive inference, DFAs, and computational complexity. In: Jantke, K.P. (ed.) AII 1989. LNCS, vol. 397, pp. 18–44. Springer, Heidelberg (1989)

[Pod74] Podnieks, K.: Comparing various concepts of function prediction. Theory of Algorithms and Programs 210, 68–81 (1974)

[RC94] Royer, J., Case, J.: Subrecursive Programming Systems: Complexity and Succinctness. In: Research monograph in Progress in Theoretical Computer Science, Birkhäuser, Boston (1994)

[Rog67] Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw Hill, New York (1967) (Reprinted by MIT Press, Cambridge, Massachusetts, 1987)

[Wie76] Wiehagen, R.: Limes-erkennung rekursiver Funktionen durch spezielle Strategien. Electronische Informationverarbeitung und Kybernetik 12, 93–99 (1976)