Strongly non-U-shaped language learning results by general techniques

John Case\textsuperscript{a}, Timo Kötzing\textsuperscript{b,*}

\textsuperscript{a} Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA
\textsuperscript{b} Hasso Plattner Institute, 14462 Potsdam, Germany

**Abstract**

In learning, a semantic or behavioral U-shape occurs when a learner first learns, then unlearns, and, finally, relearns, some target concept.

This paper introduces two general techniques and applies them especially to syntactic U-shapes in learning: one technique to show when they are necessary and one to show when they are unnecessary. The technique for the former is very general and applicable to a much wider range of learning criteria. It employs so-called self-learning classes of languages which are shown to characterize completely one criterion learning more than another.

We apply these techniques to show that, for set-driven and rearrangement-independent learning, any kind of U-shapes is unnecessary. Furthermore, we show that U-shapes are necessary in a strong way for iterative learning, contrasting with an earlier result by Case and Moelius that semantic U-shapes are unnecessary for iterative learning.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

In Section 1.1 we explain U-shaped learning. In Section 1.2 we briefly discuss the general techniques of the present paper and summarize in Section 1.3 our applications of these techniques regarding the necessity of U-shaped learning.

1.1. U-shaped learning

U-shaped learning occurs when a learner first learns a correct behavior, then abandons that correct behavior and finally returns to it once again. This pattern of learning has been observed by cognitive and developmental psychologists in a variety of child development phenomena, such as language learning, understanding of temperature, weight conservation, object permanence and face recognition. The case of language acquisition is paradigmatic. For example, a child first uses *spoke*, the correct past tense of the irregular verb to *speak*. Then the child ostensibly overregularizes incorrectly using *spaked*. Lastly and finally the child returns to using *spoke*. The language acquisition case of U-shaped learning behavior has figured prominently in cognitive science.

While the prior cognitive science literature on U-shaped learning was typically concerned with modeling how humans achieve U-shaped behavior, the papers [3,11] are motivated by the question of why humans exhibit this seemingly inefficient behavior. Is it a mere harmless evolutionary inefficiency or is it necessary for full human learning power? A technically

* Corresponding author.

E-mail addresses: case@udel.edu (J. Case), timo.koetzing@hpi.de (T. Kötzing).

http://dx.doi.org/10.1016/j.ic.2016.06.015
0890-5401/© 2016 Elsevier Inc. All rights reserved.
answer version of this question is: are there some formal learning classes of tasks for which U-shaped behavior is logically necessary? We first need to describe some formal criteria of successful learning.

An algorithmic learning function $h$ is, in effect, fed an infinite sequence consisting of the elements of a (formal) language $L$ in arbitrary order with possibly some pause symbols # in between elements. During this process, $h$ outputs a corresponding sequence $p(0), p(1), \ldots$ of hypotheses (grammars) which may generate the language $L$ to be learned. A fundamental criterion of successful learning of a language is called explanatory learning (TxtEx-learning, also called learning in the limit) and was introduced by Gold [27]. Explanatory learning requires that the learner's output conjectures stabilize in the limit to a single conjecture (grammar/program, description/explanation) that generates the input language. Behaviorally correct learning [18,33] requires, for successful learning, convergence in the limit to a sequence of correct (but possibly syntactically distinct) conjectures. Another interesting class of criteria features vacillatory learning [10,28]. This paradigm involves learning criteria which allow the learner to vacillate in the limit between at most some bounded, finite number of syntactically distinct but correct conjectures. For each criterion that we consider above (and below), a non-U-shaped learner is naturally modeled as a learner that never returns to a previously semantically abandoned correct conjecture on languages it learns according to that criterion.

Ref. [3] showed that every TxtEx-learnable class of languages is TxtEx-learnable by a non-U-shaped learner, that is, for TxtEx-learnability, U-shaped learning is not necessary. Furthermore, based on a proof in [24,3] noted that, by contrast, for behaviorally correct learning [23,18,33], U-shaped learning is necessary for full learning power. In [11] it is shown that, for non-trivial vacillatory learning, U-shaped learning is again necessary (for full learning power). Thus, in many contexts, seemingly inefficient U-shaped learning can actually increase one's learning power.

What turns out to be a variant of non-U-shaped learning is strongly non-U-shaped learning essentially defined in [43], where the learner is required never to syntactically abandon a correct conjecture on languages it learns according to that criterion. Clearly, strong non-U-shaped learnability implies non-U-shaped learnability. In our experience, for theoretical purposes, it is frequently easier to show non-U-shaped learnability by showing strong non-U-shaped learnability. Herein we especially study strong non-U-shaped learnability.

1.2. Presented techniques

The present paper presents two general techniques to tackle problems regarding U-shaped learning.

The first general technique can be used to show the necessity of U-shapes and employs so-called self-learning classes of languages. These are explained in Section 3 below. These self-learning classes of languages provide a (provably) most general way for finding classes of languages that separate two learning criteria, i.e., they give a general way of finding an example class of languages learnable with a given learning criterion, but not with another. Theorem 3.6 implies that its presented self-learning classes necessarily separate two learnability sets — iff any class does. This technique is not specialized only to analyze U-shaped learning, but can be applied to other learning criteria as well. The technique is developed and discussed further in Section 3.

The second general technique is used to show that syntactic U-shapes are unnecessary and is phrased as a characterization of strongly non-U-shaped learnability of classes of languages (Theorem 4.4).

1.3. Applications of general techniques

A learning machine is set-driven [42,37,26,28] (respectively, rearrangement-independent [37,26,28]) iff, at any time, its output conjecture depends only on the set of non-pause data it has seen (respectively, set of non-pause data and data-sequence length), not on the order of that data's presentation. Child language learning may be insensitive to the order or timing of data presentation; set-drivenness and rearrangement independence, Sd and Ri, respectively, provide two local notions of such insensitivity [10]. It is interesting, then, to see the interaction of these notions with forbidding U-shapes of one kind or another. As we shall see in Section 5, Theorems 5.2 and 5.3, proved with the aid of a general technique from Section 4, imply, for these local data order insensitivity notions, for TxtEx-learning, U-shapes, even in the strong sense are unnecessary.

An iterative learner outputs its conjectures only on the basis of its immediately prior conjecture (if any) and its current datum. As we shall see in Section 5, iterative learning provides a (first) example of a setting in which non-U-shaped and strongly non-U-shaped learning are extensionally distinct: [19] shows semantic U-shapes to be unnecessary for iterative learning, while Theorem 5.7 in the present paper implies that they are in the strong sense necessary. To prove this latter result, we actually modify the self-learning class of languages from Theorem 3.6 to make it easier to work with — although the original version must work too (by Theorems 3.6 and 5.7).

---

1. Wiehagen actually used the term semantically finite in place of strongly non-U-shaped. However, there is a clear connection between this notion and that of non-U-shapedness. Our choice of terminology is meant to expose this connection. See also [21].
2. For non-U-shaped learning, the learner (on the way to success) must not semantically abandon a correct conjecture. In general, semantic change of conjecture is not algorithmically detectable, but syntactic change is. However, in the cognitive science lab we can many times see a behavioral/syntactic change, but it is beyond the current state of the art to see, for example, grammars in people's heads — we can't yet see mere syntactic changes in people's heads.
3. Some recent papers [13,15,16,30] have also employed (different) self-learning classes for separations.
Some of our proofs involve subtle infinitary program self-reference arguments employing (variants of) the Case’s Operator Recursion Theorem (ORT) from [8,9,28,32].

Note that the present paper is an extended version of [14].

1.4. Open problems

Some problems regarding the necessity of U-shapes of one kind or another still remain open. An iterative with counter learner is an iterative learner which, in making a conjecture, also has access to the data-sequence length so far. For example, it is still open whether semantic U-shapes are necessary for iterative with counter learning — as asked in [19]. If so, then the relevant self-learning class from Theorem 3.6 below must provide a separation.

See the end of Section 5 for some more open problems regarding the necessity of any one of the two kinds of U-shapes for learning criteria of the present paper.

2. Mathematical preliminaries

Unintroduced complexity theoretic notation follows [35]. Other unintroduced notation follows [36].

By \( \mathbb{N} \) we denote the set of natural numbers, \( \{0, 1, 2, \ldots \} \). We let \( \mathbb{N}^+ = \mathbb{N} \setminus \{0\} \). The symbols \( \subseteq, \subseteq, \supseteq, \supseteq \) respectively denote the subset, proper subset, superset and proper superset relation between sets. For any set \( A \), we let \( \text{Pow}(A) \) denote the set of all subsets of \( A \). We let \( \Delta \) denote the symmetric difference of two sets. \( \emptyset \) denotes both the empty set and the empty sequence. \( \mathcal{P} \) denotes the set of all partial functions \( \mathbb{N} \to \mathbb{N} \).

The quantifier \( V^x \) means “for all but finitely many \( x \)”, the quantifier \( \exists^x \) means “for infinitely many \( x \).”

With dom and range we denote, respectively, domain and range of a given function. We sometimes denote a partial function \( f \) of \( n \geq 0 \) 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. The composition of two functions \( f, g \) is denoted by \( f \circ g \).

Whenever we consider tuples of natural numbers as input to a function, it is understood that they are suitably encoded as a single natural number; similarly for finite sets and sequences, see, for example, [35] for details.

If a function \( f \) is not defined for some argument \( x \), then we denote this fact by \( f(x)^{\uparrow} \), and we say that \( f \) on \( x \) diverges; the opposite is denoted by \( f(x)^{\downarrow} \), and we say that \( f \) on \( x \) converges. If \( f \) on \( x \) converges to \( p \), then we denote this fact by \( f(x)^{\downarrow} = p \).

The special symbol \( ? \) is used as a possible hypothesis (meaning “no change of hypothesis”): note that typically the learner can remember its past conjecture and is thus able to output this past hypothesis instead of \( ? \). We write \( f \to p \) to denote that \( f : \mathbb{N} \to \mathbb{N} \cup \{?)\} \) converges to \( p \), i.e., \( \exists q : f(q) = p \land \forall x \geq q : f(x)^{\downarrow} \in \{?, p\} \). \( \mathcal{P} \) and \( \mathcal{R} \) denote, respectively, the set of all partial and the set of all computable functions (mapping \( \mathbb{N} \to \mathbb{N} \)).

We let \( \varphi \) be any fixed acceptable programming system for \( \mathcal{P} \) ([36], an acceptable programming system could, for example, be based on a natural programming language such as C or Java, or on Turing machines). Further, we let \( \varphi_p \) denote the partial computable function computed by the \( \varphi \)-program with code number \( p \). A set \( L \subseteq \mathbb{N} \) is computably enumerable (c.e) iff it is the domain of a partial computable function. Let \( \mathcal{E} \) denote the set of all c.e sets. We call a set computable iff its characteristic function is computable. We let \( W \) be the mapping such that \( \forall e : W(e) = \text{dom}(\varphi_e) \). For each \( e \), we write \( W(e) \) instead of \( W(e) \). \( W \) is, then, a mapping from \( \mathbb{N} \) onto \( \mathcal{E} \). We say that \( e \) is an index, or program, (in \( W \)) for \( W_e \).

The symbol \# is pronounced pause and is used to symbolize “no new input data”. For each (possibly infinite) sequence \( q \) with its range contained in \( \mathbb{N} \cup \{?)\} \), let \( \text{content}(q) = (\text{range}(q) \setminus \{?)\}) \). For any function \( f \in \mathcal{P} \) and all \( i \), we use \( f[i] \) to denote the sequence \( f(0)^{\downarrow}, \ldots, f(i-1)^{\downarrow} \) (the empty sequence if \( i = 0 \) and undefined, if one of these values is undefined).

We will make use of a padded variant of the s-m-n Theorem [36]. Intuitively, s-m-n permits algorithmic storage of arbitrary data (and, hence, programs) inside any program. The suitable padded variant of s-m-n we use herein states that there is a strictly monotonic increasing computable function \( s \) such that

\[
\forall a, b, c : \varphi_{s(a,b)}(c) = \varphi_{s(b,c)}.
\]

In (1), \( \varphi \)-program \( s(a, b) \) is essentially \( \varphi \)-program \( a \) with datum \( b \) stored inside. We will also use a suitably padded version of Case’s Operator Recursion Theorem (ORT), providing infinitary self (and other) reference [8,9,28]. See [36] for a treatment of computable operators (also called recursive operators). ORT itself states that, for all computable operators \( \Theta : \mathcal{P} \to \mathcal{P} \),

\[
\exists e \in \mathcal{R} \forall a, b : \varphi_{e(a)}(b) = \Theta(e)(a, b).
\]

In the padded version we employ, the function \( e \) will also be strictly monotone increasing.

2.1. Computability-theoretic learning

In this section, we formally define several criteria for computability-theoretic learning.

\footnote{If \( f \) on \( x \) converges should not be confused with \( f \) converges to.}
In this section we formally introduce our setting of learning in the limit and associated learning criteria. We follow [29] in its “building-blocks” approach for defining learning criteria.

A learner is a partial function from \( \mathbb{N} \) to \( \mathbb{N} \cup \{\#\} \) (note that, for this paper, we do not always require computability of learners). A language is a subset \( L \subseteq \mathbb{N} \). Any total function \( T : \mathbb{N} \to \mathbb{N} \cup \{\#\} \) is called a text. Note that the only text for the empty language is an infinite sequence of \#'. For any given language \( L \), a text \( T \) for \( L \) is a text \( T \) such that \( \text{content}(T) = L \). This kind of text is what learners usually get as information; we assume that the \#-symbol and the data are appropriately coded to serve as input to (partial) computable functions. With \( \text{Txt}(L) \) we denote the set of all texts for \( L \).

A sequence generating operator is an operator \( \beta \) taking as arguments a function \( h \) (the learner) and a text \( T \) and outputting a one-argument function. For \( p \) the output of \( \beta \) on \( h \) and \( T \) we call \( p \) the learning sequence of \( h \) given \( T \). Intuitively, \( \beta \) defines how a learner can interact with a given text to produce a sequence \( p \) of conjectures.

We define the sequence generating operators \( G \) and \( \text{It} \) (corresponding to the learning criteria discussed in the introduction) as follows. For all learners \( h \), texts \( T \) and all \( i \),

\[
G(h, T)(i) = h(T[i]);
\]

\[
\text{It}(h, T)(i) = \begin{cases} h(\emptyset), & \text{if } i = 0; \\ h(\text{It}(h, T)(i-1), T(i-1)), & \text{otherwise.} \end{cases}
\]

Thus, in iterative learning, the learner has access to the previous conjecture and the current datum, but not directly to any strictly previous data as in \( G \)-learning.

Another interesting sequence generating operator is set-driven learning ([42], denoted \( \text{Sd} \)). We let, for all learners \( h \), texts \( T \) and all \( i \),

\[
\text{Sd}(h, T)(i) = h(\text{content}(T[i]), i).
\]

A slight variant of set-driven learning, called rearrangement-independent learning additionally provides information about the length of the input sequence. We let, for all learners \( h \), texts \( T \) and all \( i \),

\[
\text{Ri}(h, T)(i) = h(\text{content}(T[i]), i).
\]

Finally, we are also interested in iterative-with-counter learning [19]; for all learners \( h \), texts \( T \) and all \( i \), we let

\[
\text{ItCr}(h, T)(i) = \begin{cases} h(\emptyset), & \text{if } i = 0; \\ h(\text{ItCr}(h, T)(i-1), T(i-1), i-1), & \text{otherwise.} \end{cases}
\]

Successful learning requires the learner to observe certain restrictions, for example convergence to a correct index or non-U-shapedness. These restrictions are formalized in our next definition.

A sequence acceptance criterion is a predicate \( \delta \) on a learning sequence and a text. We give the examples of explanatory (\( \text{Ex} \)) non-U-shaped (\( \text{NU} \)) and strongly non-U-shaped (\( \text{SNU} \)) learning, which were discussed in Section 1. Formally, we define, for all conjecture sequences \( p \) and texts \( T \),

\[
\text{Ex}(p, T) \iff [\exists q : \text{content}(T) = W_q \land \forall i : p(i) = q];
\]

\[
\text{NU}(p, T) \iff [\forall i : p(i) = W_p(i) \Rightarrow \text{content}(T) \Rightarrow W_p(i) = W_p(j)];
\]

\[
\text{SNU}(p, T) \iff [\forall i, j, k : i \leq j \leq k \land W_p(i) = W_p(k) = \text{content}(T) \Rightarrow p(i) = p(j)].
\]

We combine any two sequence acceptance criteria \( \delta \) and \( \delta' \) by intersecting them; we denote this by juxtaposition (for example, \( \text{NU} \) is meant to be always used together with \( \text{Ex} \)).

For any set of allowed learners \( \mathcal{C} \subseteq \mathcal{P} \), sequence generating operator \( \beta \) and any sequence acceptance restriction \( \delta \), \((\mathcal{C}, \beta, \delta)\) is a learning criterion. We also write this learning criterion as \( \text{Ctxt}\beta\delta \) and omit \( \mathcal{C} \) if \( \mathcal{C} = \mathcal{P} \). A learner \( h \) \( \text{Ctxt}\beta\delta \)-learns the empty set if \( h \notin \mathcal{C} \) and otherwise the set

\[
\text{Ctxt}\beta\delta(h) = \{ L \in \mathcal{L} \mid \forall T \in \text{Txt}(L) : \delta(h(T), T) \}
\]

With \( \{\text{Ctxt}\beta\delta\} \) we denote the collection of all sets of language \( \mathcal{C} \) such that there is a learner \( h \) with \( \mathcal{C} \subseteq \text{Ctxt}\beta\delta(h) \).

With these definitions we see that the informally introduced criterion \( \text{TxtEx} \) from the introduction corresponds to \( \text{TxtGEx} \).

\[5\ h(\emptyset) \text{ denotes the initial conjecture made by } h.\]
Starred Learners
For two sequence generating operators $\beta$ and $\beta'$ we write $\beta \leq \beta'$ if, for all $h$ there is $h'$ with

$$\forall T : \beta(h, T) = \beta'(h', T).$$

Intuitively, any learners learning with information corresponding to $\beta$ can be turned into learners learning according to information from $\beta'$. Clearly, for all $\beta$ give above, we have $\beta \leq \mathcal{G}$. Thus, for any $\beta$-learner $h$, we let $h^*$ be the corresponding $\mathcal{G}$-learner. That is, for all sequences $\sigma$, $h^*(\sigma)$ denotes the current conjecture of $h$ after being fed the sequence $\sigma$ of data items.

In particular, for $h \in \mathcal{P}$ and $\sigma$ a sequence, we have the following.

- If $h$ is a set-driven learner:
  $$h^*(\sigma) = h(\text{content}(\sigma)).$$

- If $h$ is a rearrangement-independent learner:
  $$h^*(\sigma) = h(\text{content}(\sigma), \text{len}(\sigma)).$$

2.2. Locking

In this section we will make a number of definitions in dependence on a learning criterion $\mathcal{C}^{\text{Txt}}_{\beta \delta}$ and a learner $h \in \mathcal{C}$.

We first define the concept of a stabilizer sequence (introduced in [2] and called “stabilizer segment” in [26]). Let $\beta$ be a sequence generating operator. Let $L$ be a language and $h \in \mathcal{P}$ a learner. A sequence $\sigma$ with elements from $L$ is said to be a $\beta$-stabilizer sequence of $h$ on $L$ if

$$(\forall T \in \mathcal{Txt}(L)|\sigma \subseteq T) \forall i \geq \text{len}(\sigma) : \beta(h, T)(\text{len}(\sigma)) = \beta(h, T)(i);$$

Intuitively, a stabilizer sequence $\sigma$ of $h$ on $L$ is a sequence of elements from $L$ such that $h$ on any text extending $\sigma$ will never make a change of conjecture after having seen $\sigma$. For any stabilizer sequence $\sigma$ of $h$ on $L$, we call $\sigma$ a locking sequence if the conjecture of $h$ after $\sigma$ is a correct conjecture.

It is well known that, if a learner $h \mathcal{C}^{\text{Txt}}_{\text{GEx}}$-learns a language $L$, then there is a stabilizer sequence of $h$ on $L$ (see [28]). However, texts do not necessarily contain such a sequence as an initial segment. Below, we define a learning restriction that requires a learner and a language to have stabilizer sequences as initial sequences of all texts for the language. This motivates us to define the following notion. We say that a learner $h \mathcal{C}^{\text{Txt}}_{\beta \delta}$-learns a set of languages $L$ stabilizingly iff $h \mathcal{C}^{\text{Txt}}_{\beta \delta}$-learns $L$ and, for all $L \in \mathcal{L}$ and $T$ a text for $L$, there is $n$ such that $T[n]$ is a $\beta$-stabilizer sequence of $h$ on $L$.

We define a $\beta$-sink of $h$ on $L$ to be a conjecture $e$ such that

$$(\forall T \in \mathcal{Txt}(L)|\sigma \subseteq T) \forall i_0 : [\beta(h, T)(i_0) = e \Rightarrow (\forall i \geq i_0)(\beta(h, T)(i) = e)].$$

Intuitively, a sink is a conjecture never abandoned on texts for $L$. A stabilizer sequence $\sigma$ for $h$ on a language $L$ is called a sink-stabilizer sequence for $h$ on $L$ if for all texts $T$ for $L$ with $\sigma \subseteq T$ we have that $\beta(h, T)(\text{len}(\sigma))$ is a $\beta$-sink of $h$ on $L$.

We say that a learner $h \mathcal{C}^{\text{Txt}}_{\beta \delta}$-learns a set of languages $L$ sink-stabilizingly iff $h \mathcal{C}^{\text{Txt}}_{\beta \delta}$-learns $L$ and, for all $L \in \mathcal{L}$ and $T$ a text for $L$, there is $n$ such that $T[n]$ is a $\beta$-sink-stabilizer sequence of $h$ on $L$.

Sink-stabilizing is of interest, as we show a characterization theorem (Theorem 4.4 below) of strongly non-U-shaped learning in terms of sink-stabilizing learning.

3. Self-learning classes of languages for separations

In this section we discuss a way of showing U-shapes to be necessary. Formally, this is done via showing that a learnability class separates from its non-U-shaped variant.

The approach described below is very general and can be applied to show separation results in many other areas of computability-theoretic learning in the limit as well.

The key idea is that of self-learning classes of languages. In the previous literature, self-describing classes of languages have been used. A particularly simple example class of self-describing languages, taken from [18, Theorem 1], is

$$\mathcal{L}_0 = \{L \text{ is computable } | L \neq \emptyset \land W_{\text{min}}L = L\}.$$ 

Intuitively, each $L \in \mathcal{L}_0$ gives a complete description of itself, encoded (as a $W$-index) within only finitely many (in fact, one) of its elements. It is well-known, using standard computability theoretic arguments, that these kinds of classes of languages are very big (for example, $\mathcal{L}_0$ contains a finite variant of any given computable language, which can easily be seen using Kleene’s Recursion Theorem).

---

6 See [28]. In there, the term “self-describing” was used on page 71 in the context of function learning and extended on page 97 to language learning.

7 A set A is a finite variant of a set B iff AΔB is a finite set.
Many variants of self-describing classes of languages have been used for separation results within computability-theoretic learning (see, for example, [2,18,22,10,28]). Showing a separation with a complicated self-describing class of languages sometimes requires a non-trivial learner (see, for extreme examples, [12]).

We now take the technique of self-describing one step further. A self-learning class of languages is such that each element of each language of the class provides instructions for what to compute and output as a new hypothesis. Thus, all a learner needs to do is to execute the instructions given by its latest datum. For example, an informal\(^8\) learner \(h_1\) can be defined such that, for all sequences \(\sigma\) and numbers \(x\),

\[
h_1(\sigma \circ x) = \phi_x(\sigma \circ x).
\]

(5)

Intuitively, \(h_1\) interprets the latest datum as a program in the \(\psi\)-system and runs this program on all known data, which is \(\sigma \circ x\). Variants of this \(h_1\) can be defined to obtain learners with special additional properties, such as totality or set-drivenness (see Theorem 3.6).

In practice, the general scheme is as follows. Suppose we want to show, for two learning criteria \(I_0\) and \(I_1\), \([I_1] \setminus [I_0] \neq \emptyset\). Then we define a simple learner, for example \(h_1\) above, and let \(L_1\) be the class of all languages \(I_1\)-learned by \(h_1\). All that would remain to be shown is that \(L_1\) is not \(I_0\)-learnable, which can often be done using ORT.

Below, in Theorem 3.6, we give a very general result regarding some self-learning classes of languages guaranteed to witness separations when they exist. In order to do so, we proceed next by making some formal definitions.

For a function \(e \in P\) and a language \(L\), we let \(e(L) = \{e(x) \mid x \in L\}\); for a class of languages \(\mathcal{L}\), we let \(e(\mathcal{L}) = \{e(L) \mid L \in \mathcal{L}\}\).

Let \(\mathcal{P}_{c_{1,1}} \subseteq P\) (respectively, \(\mathcal{R}_{c_{1,1}} \subseteq R\)), denote the set of all 1-1 partial (respectively, total) computable functions with a computable domain and range.

**Definition 3.1.** A learning criterion \(I\) is called computably \(\mathcal{P}_{c_{1,1}}\)-invariant iff there is a computable operator \(\Theta : \mathcal{P}^2 \to \mathcal{P}\) such that

\[
\forall h \in P, \forall e \in \mathcal{P}_{c_{1,1}} : e(I(h)) \subseteq I(\Theta(h, e)).
\]

(6)

Intuitively, if a learner \(h\) learns languages \(L\), then \(\Theta(h, e)\) learns \(e(L)\). Note that this property was called robustness in previous publications [15,30].

**Remark 3.2.** Let \(I\) be computably \(\mathcal{P}_{c_{1,1}}\)-invariant. Then we have

\[
\forall e \in \mathcal{P}_{c_{1,1}}, \forall \mathcal{L} \subseteq L \in [I] \Leftrightarrow e(\mathcal{L}) \in [I].
\]

**Remark 3.3.** For any of the choices \(\mathcal{C} \in \{R, P\}, \beta \in \{G, R_i, Sd, ItCtr, It\}\) and \(\delta \in \{SNUEx, NUEx, Ex\}\) we have that \((\mathcal{C}, \beta, \delta)\) is computably \(\mathcal{P}_{c_{1,1}}\)-invariant.

**Proof.** Let \(\mathcal{L}\) be \(C Txt\beta\delta\)-learnable as witnessed by \(h\) and let \(e \in \mathcal{P}_{c_{1,1}}\). We show \(e(\mathcal{L})\) is \(C Txt\beta\delta\)-learnable for the examples of \(\beta \in \{G, It\}\), all other cases are similar. Using padded \(s\)-m-n, there is a strictly monotone increasing \(f \in R\) such that

\[
\forall p : W_f(p) = e(W_p).
\]

Regarding \(G\): Let \(h' \in P\) be such that

\[
\forall \sigma : h'(\sigma) = \begin{cases} f(h(e^{-1} \circ \sigma)), & \text{if content}(\sigma) \subseteq \text{range}(e), \quad \text{\textbullet} \\ 0, & \text{otherwise}. \end{cases}
\]

Clearly, \(e(\mathcal{L}) \subseteq C TxtGEx(h')\); it is easy to verify that (strongly) non-U-shapedness is preserved.

Regarding \(It\): Let \(h' \in P\) be such that

\[
\forall p, x : h'(p, x) = \begin{cases} f(h(f^{-1}(p), e^{-1}(x))), & \text{if } p \in \text{range}(f) \text{ and } x \in \text{range}(e); \\ 0, & \text{otherwise}. \end{cases}
\]

The remaining reasoning for \(It\) is analogous to that for \(G\). \(\square\)

**Definition 3.4.** Let \(I = (C, \beta, \delta)\) be a learning criterion. We call \(I\) data normal iff, for all \(p_0\) such that \(W_{p_0} = \emptyset\), there is a computable operator \(\hat{\Theta} : \mathcal{P} \to \mathcal{P}\) such that

\[
I(h) \subseteq \hat{I}(\hat{\Theta}(h))
\]

(7)

and (a)-(d) below.

---

\(^8\) Note that we ignore the possible input of \(x = \#\).

\(^9\) Note that we use the convention \(f(\#) = \#\) and \(e(\#) = \#\) for ease of notation.
(a) There is an \( f_\beta \in \mathcal{R} \) such that, for all learners \( h \), texts \( T \) and all \( i > 0 \),
\[
\beta(h, T)(i) = h(f_\beta(T[i], \beta(h, T)[i])). \tag{8}
\]
(b) There is a function \( d_\beta \in \mathcal{R} \) such that, for all texts \( T \) and all \( i \), if we have \( \beta(\hat{\Theta}(h), T)[i] \downarrow \), then
\[
d_\beta(f_\beta(T[i], \beta(\hat{\Theta}(h), T)[i])) \in \begin{cases} 
\# & \text{if content}(T[i]) = \emptyset; \\
\text{content}(T[i]) & \text{otherwise}. \end{cases} \tag{9}
\]
(c) For all \( h \in \mathcal{P} \),
\[
\forall \sigma, \tau : d_\beta(f_\beta(\sigma, \tau)) = \# \Rightarrow \hat{\Theta}(h)(f_\beta(\sigma, \tau)) = p_0. \tag{10}
\]
(d) For all \( h, h' \in \mathcal{P} \),
\[
[\forall L \in I(h) \forall T \in \text{Txt}(L) : \beta(h, T) = \beta(h', T)] \Rightarrow I(h) \subseteq I(h'). \tag{11}
\]

**Remark 3.5.** For any of the choices \( \mathcal{C} \in \{ \mathcal{R}, \mathcal{P} \} \), \( \beta \in \{ \mathcal{G}, \mathcal{Ri}, \mathcal{Sd}, \mathcal{ItCtr}, \mathcal{It} \} \) and \( \delta \in \{ \text{SNUEx, NUEX, Ex} \} \) we have that \((\mathcal{C}, \beta, \delta)\) is data normal.

**Proof.** Regarding (a), we can, for example, take, for each \( \sigma, \tau \),
\[
\begin{align*}
\mathcal{G}_\beta(\sigma, \tau) &= \sigma; \\
\mathcal{Sd}_\beta(\sigma, \tau) &= \text{content}(\sigma); \\
\mathcal{It}_\beta(\sigma, \tau) &= \begin{cases} 
(\text{last}(\tau), \text{last}(\sigma)) & \text{if } \sigma \neq \emptyset \text{ and } \tau \neq \emptyset; \\
0 & \text{otherwise}.
\end{cases}
\end{align*}
\]
These work for each \( \mathcal{C} \in \{ \mathcal{R}, \mathcal{P} \} \). Similarly, we can get \( f_\beta \) for the other relevant \( \beta \).

We show Requirements (b) and (c) for \( \beta = \mathcal{It} \); all other cases are similar, but easier.

Let \( p_0 \) be such that \( W_{p_0} = \emptyset \). By the padded s-m-n Theorem, there is a strictly monotone increasing \( z \in \mathcal{R} \) such that
\[
\forall e, a : W_{z(e, a)} = W_e \quad \text{and} \quad p_0 \notin \text{range}(z).
\]
Then there is a \( \gamma \in \mathcal{R} \) such that \( \forall e, a : y(z(e, a)) = a \). Let \( \hat{\Theta} \) be such that
\[
\forall h, e, x : \hat{\Theta}(h)(e, x) = \begin{cases} 
p_0 & \text{if } x = \# \text{ and } e \in \{ ?, p_0 \}; \\
z(h(?, x), x) & \text{else if } e \notin \text{range}(z) \\
z(h(e', x), a) & \text{otherwise, with } e = z(e', a).
\end{cases}
\]

Let \( d \in \mathcal{R} \) such that
\[
\forall e, x : d(e, x) = \begin{cases} 
x & \text{if } x \neq \#; \\
\# & \text{else if } e \notin \text{range}(z); \\
a & \text{otherwise, with } e = z(e', a).
\end{cases}
\]

Then \( d \) fulfills (b), and (c) is obvious.

Requirement (d) is clear for all criteria considered. \( \Box \)

Next (Theorem 3.6) is the main result of this section, giving sufficient conditions for when a separation will necessarily be witnessed by a specific self-learning class of languages. As a corollary, we get that for each pair of learning criteria with components from \( \{ \mathcal{R}, \mathcal{P} \} \), \( \{ \mathcal{G}, \mathcal{Ri}, \mathcal{Sd}, \mathcal{ItCtr}, \mathcal{It} \} \) and \( \{ \text{SNUEx, NUEX, Ex} \} \), any separations are witnessed by such classes! In this sense, self-learning classes of languages capture the essence of separations (when they exist). Note that the proof of the theorem would simplify a lot, were one to suppose somewhat stronger properties of the learning criteria, in particular, excluding the use of \( \mathcal{It} \) and \( \mathcal{ItCtr} \) as sequence generating operators. Theorem 3.6 can be modified to cover other kinds of criteria, for example, those pertaining to learnability by total learners. In a future paper, we will analyze self-learning-classes in more depth and will provide further theorems like Theorem 3.6.

---

10 Intuitively, the \( i \)-th conjecture of \( h \) on \( T \) depends only on some information (as specified by \( f_\beta \)) about the first \( i \) datapoints and conjectures.

11 Intuitively, from the information given by \( f_\beta \), a datum (if any) that this datum is based on can be extracted.

12 Intuitively, constantly outputting one and the same index for the empty language is a viable strategy as long as no numerical data has been presented.

13 Intuitively, changing a learner on inputs that do not present data from a language learned does not harm learnability.
**Theorem 3.6.** Let \( I_0 \) and \( I_1 \) be computably \( \mathcal{P}_{c1,1} \)-invariant learning criteria. Suppose \( I_1 \) is data normal as witnessed by \( f_1 \) and \( d_1 \). Let \( p_0 \) be such that \( W_{p_0} = \emptyset \) and \( h_1 \) be such that

\[
\forall x : h_1(x) = \begin{cases} p_0, & \text{if } d_1(x) = \#; \\
\varphi_{d_1(x)}(x), & \text{otherwise.} \end{cases}
\]

Further, let \( L_1 = I_1(h_1) \). Then we have

\[
[I_1 \setminus I_0] \neq \emptyset \Rightarrow L_1 \not\subseteq I_0.
\]

In other words, if \( I_1 \) allows for learning a set of languages which is not \( I_0 \)-learnable, then \( L_1 \) is an example such class.

**Proof.** The implication “\( \Leftarrow \)’” is obvious. Regarding “\( \Rightarrow \)’”, let \( L \) be \( I_1 \)-learnable as witnessed by \( h \) and suppose \( L \) is not \( I_0 \)-learnable. Let \( \Theta \) be as given by \( I_1 \) being computably \( \mathcal{P}_{c1,1} \)-invariant. Let \( \hat{\Theta} \) be as given by \( I_1 \) being data normal. By padded \( \text{ORT} \), there is a strictly monotone increasing \( e \in \mathcal{R} \) such that

\[
\forall x, y : \varphi_{e(x)}(y) = (\hat{\Theta} \circ \Theta)(h, e)(y).
\]

As \( e \in \mathcal{P}_{c1,1} \) and \( I_0 \) is computably \( \mathcal{P}_{c1,1} \)-invariant, we have, from Remark 3.2 with \( I_0 \) in the place of \( I, e(L) \notin [I_0] \). It now suffices to show \( e(L) \subseteq L_1 \), as this would imply \( L_1 \not\subseteq [I_0] \) as desired.

Suppose \( I_1 = \{ \hat{C}_1, \hat{\sigma}_1, \hat{\delta}_1 \} \). Let \( L \in e(L) \) and \( T \in \text{Txt}(L) \). We show, by induction on \( i \),

\[
\forall i : \beta_1(h_1, T)(i) = \beta_1((\hat{\Theta} \circ \Theta)(h, e), T)(i).
\]

Let \( h' = \Theta(h, e) \). For all \( i \) with content\( (T[i]) = \emptyset \), we have

\[
\beta_1(h_1, T)[i] = h_1(f_1(T[i], \beta_1(h_1, T)[i])) = p_0
\]

\[
= \hat{\Theta}(h') \cdot f_1(T[i], \beta_1(h', T)(i))
\]

\[
= \beta_1(h', T)(i).
\]

Let \( i \in \mathbb{N} \), suppose \( \text{content}(T[i]) \neq \emptyset \) and (inductively) \( \beta_1(h_1, T)[i] = \beta_1((\hat{\Theta} \circ \Theta)(h, e), T)[i] \). Let \( x = f_1(T[i], \beta_1(h_1, T)[i]) = f_1(T[i], \beta_1((\hat{\Theta} \circ \Theta)(h, e), T)[i]) \).

Note that \( d_1(x) \in \text{content}(T[i]) \subseteq L \subseteq \text{range}(e) \). We have

\[
\beta_1(h_1, T)(i) = h_1(x) \equiv \varphi_{d_1(x)}(x) \equiv (\hat{\Theta} \circ \Theta)(h, e)(x)
\]

\[
= \beta_1((\hat{\Theta} \circ \Theta)(h, e), T)(i).
\]

This concludes the induction. Thus, \( h_1 \) on any text for a language from \( e(L) \) makes the same conjectures as \( (\hat{\Theta} \circ \Theta)(h, e) \) on \( T \). By (6) and (7), \( (\hat{\Theta} \circ \Theta)(h, e) \) \( I_1 \)-learns \( e(L) \); thus, using (d) of \( I_1 \) being data normal, \( e(L) \subseteq I_1(h_1) = L_1 \).

4. **Helping remove U-shapes**

In this section we provide, in Theorem 4.4, a general technique for helping with the removal of U-shapes from a learner, *preserving what is learned*. When applicable, this shows U-shapes unnecessary. The main result of this section is Theorem 4.4, which characterizes strongly non-U-shaped learning in terms of sink-stabilizing learning; this theorem is applied in Section 5.

**Lemma 4.1.** Let \( h \in \mathcal{P} \). Then there is an infinite set \( L \in \mathcal{E} \) such that \( h \) does not \( \text{TxtGEx} \)-learn any \( L' \supseteq L \) such that \( L' \setminus L \) is finite.

**Proof.** Trivial if \( \mathbb{N} \notin \text{TxtGEx}(h) \). Otherwise, let \( \sigma \) be a locking sequence for \( h \) on \( \mathbb{N} \), \( D = \text{content}(\sigma) \). Then, obviously, \( h \) does not learn any \( L \) such that \( D \subseteq L \subseteq \mathbb{N} \). In this case, then, let \( L \) be an infinite set \( c \in \mathcal{E} \) set with infinite complement such that \( D \subseteq L \subseteq \mathbb{N} \). Clearly this \( L \) suffices. \( \square \)

For each \( h \in \mathcal{P} \), let \( L_h \) denote a set \( L \) corresponding to \( h \) and as shown existent in Lemma 4.1.

Let \( h \in \mathcal{P}, L = \text{TxtGEx}(h) \) and let \( Q \) be a \( c_{<} \) predicate on two arguments. Then, using padded \( s \)-\( m \)-\( n \), there is a strictly monotone increasing function \( p_{h,q} \in \mathcal{R} \) with, for all \( e, x \),

\[
W_{p_{h,q}}(e,x) = \{ y \in L_h \mid Q(e, x) \} \cup \{ y \in W_e \mid \text{not } (Q(e, x) \text{ in } \leq y \text{ steps}) \}.
\]

---

\[14\] This version of the proof is due to Frank Stephan [40].
Lemma 4.2. Let \( h \in \mathcal{P} \), \( \mathcal{L} = \text{TxtGEx}(h) \) and \( Q \) be a \( c \in \mathcal{P} \) set. For all \( e, x \), we have
\[
W_{\rho_{h,Q}(e,x)}(e,x) \in \mathcal{L} \Rightarrow \neg Q(e,x) \\
\Rightarrow W_{\rho_{h,Q}(e,x)} = W_e.
\] (15)

Proof. Immediate from the choice of \( L_h \). \( \square \)

Let \( \beta \in \{ \mathbf{G}, \mathbf{Ri}, \mathbf{Sd}, \mathbf{ItCtr}, \mathbf{It} \} \). We have that
\[
\{ (e_0, e_1, e_2) \mid e_0 \text{ is not a } \beta \text{-sink of } \varphi_{e_1, e_2} \text{ on } W_e \}
\]
is \( c \in \mathcal{P} \).

Remember that \( \mathcal{R}_{c1-1} \subseteq \mathcal{R} \) denotes the set of all 1-1 total computable functions with computable domain and range.

Definition 4.3. A sequence generating operator \( \beta \) is called 1-1 \textit{left-modifiable} iff
\[
\forall r \in \mathcal{R}_{c1-1} \exists s_l, s_r \in \mathcal{P}, T \in \text{Txt} : \beta(s_l \circ h \circ s_r, T) = r \circ \beta(h, T).
\]

Note that \( \mathbf{G}, \mathbf{Ri}, \mathbf{Sd}, \mathbf{ItCtr} \text{ and } \mathbf{It} \) are 1-1 left-modifiable: for any given \( \beta \in \{ \mathbf{G}, \mathbf{Ri}, \mathbf{Sd} \} \) and given \( r \in \mathcal{R}_{c1-1} \) we can choose \( s_l = r \) and \( s_r \) the identity; for the two variants of iterative learning we similarly use \( s_l = r \) but now define employ \( s_r \) to decode the previous conjecture.

We now present the main theorem of this section, characterizing strongly non-U-shaped learning.

Theorem 4.4. Let \( \beta \leq \mathbf{G} \) be 1-1 left-modifiable. Let \( \mathcal{C} \in \{ \mathcal{P}, \mathcal{R} \} \) and let \( \mathcal{L} \subseteq \mathcal{R} \). Then the following are equivalent.

(i) \( \mathcal{L} \) is \( C\text{Txt}/\beta\text{SNUEx}-\text{learnable}. \\
(ii) \( \mathcal{L} \) is sink-stabilizingly \( C\text{Txt}/\beta\text{Ex}-\text{learnable}. \\

Proof. “(i) \Rightarrow (ii)”: Let \( h_0 \in \mathcal{C} \) be a \( \text{Txt}/\beta\text{SNUEx}-\text{learner} \) for \( \mathcal{L} \). Let \( L \in \mathcal{L} \) and let \( T \) be a text for \( L \). Let \( k \) be such that \( h_0 \) has converged on \( T \) after \( T[k] \) to some \( e \). Then
\[
W_e = L.
\] (17)

To show that \( e \) is a \( \beta \)-sink of \( h_0 \) on \( L \): Let \( T \in \text{Txt}(L) \) and \( i_0 \) be given such that \( \beta(h_0, T)(i_0) = e \). Let \( i \geq i_0 \). Then, as \( h_0 \) is strongly non-U-shaped on \( L \) and from (17), \( \beta(h_0, T)(i) = e \).

Regarding “(ii) \Rightarrow (i)”: Let \( h_0 \in \mathcal{C} \), be a sink-stabilizing \( \text{Txt}/\beta\text{Ex}-\text{learner} \) for \( \mathcal{L} \). Let \( Q \) be a \( c \in \mathcal{P} \) predicate such that
\[
\forall e : Q(e) \iff e \text{ is not a } \beta \text{-sink of } h_0 \text{ on } W_e.
\] (18)

Recall the definition of starred learners from Section 2.1, which uses \( \beta \leq \mathbf{G} \). Let \( p = p_{h_0, Q} \). Let \( \beta \)'s left-modifiability with respect to \( p \) be witnessed by \( s_l \) and \( s_r \in \mathcal{R} \). Let \( h \in \mathcal{R} \) be such that
\[
h = s_l \circ h_0 \circ s_r.
\]

Note that, if \( h_0 \in \mathcal{R} \), then \( h \in \mathcal{R} \).

Claim 1. \( h \) is strongly non-U-shaped.

Proof of Claim 1. Let \( L \in \mathcal{L} \), \( e \in \mathbb{N} \) and \( \sigma \) such that content(\( \sigma \)) \( \subseteq L \). Suppose \( h^*(\sigma) = p(e) \) and
\[
W_{p(e)}(e) \in L \in \mathcal{L}.
\]

From (15) we get \( \neg Q(e) \). Hence, from the definition of \( Q \) in (18), for all texts \( T \) for \( L \) extending \( \sigma \), we have that \( h_0 \) has syntactically converged after seeing \( \sigma \). Thus, \( h \) has syntactically converged after seeing \( \sigma \) (and, thus, does not exhibit a U-shape). \( \square \) (FOR CLAIM 1)

Claim 2. \( h \) \( \text{Txt}/\beta\text{Ex}-\text{learns} \mathcal{L} \).

Proof of Claim 2. Let \( L \in \mathcal{L} \) and let \( T \) be a text for \( L \). As \( h_0 \) is sink-stabilizing, there is \( k \) minimal such that, with \( e = h_0^*(T[k]) \),
\[
e \text{ is a sink of } h \text{ on } L.
\] (19)
Thus, \( h_0 \) converges on \( T \) to \( e \); therefore,

\[
W_e = L. \tag{20}
\]

Furthermore, \( h \) on \( T \) converges to \( p(e) \) and, by \( (19) \), \( \neg Q(e) \); hence,

\[
W_{p(e)} = W_e = L. \tag{16} \tag{20} \]  \( \square \) FOR CLAIM 2 \( \square \)

5. Applications of the techniques

In this section we essentially apply general techniques presented in the two just previous sections to prove a number of results pertaining to the necessity of U-shapes in learning.

The following lemma will be useful for later proofs. Parts (i) and (iii) have been shown in [26].

**Lemma 5.1.** Let \( h \in \mathcal{P} \) and let \( C \in \{ \mathcal{P}, \mathcal{R} \} \). Then

(i) Any \( \mathcal{C} \mathcal{T} \mathcal{x} \mathcal{G} \mathcal{E} \mathcal{x} \)-learnable set can be so learned stabilizingly;

(ii) Any \( \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x} \)-learner is stabilizing;

(iii) Any \( \mathcal{C} \mathcal{T} \mathcal{x} \mathcal{R} \mathcal{I} \mathcal{E} \mathcal{x} \)-learnable set can be so learned stabilizingly.

**Proof.** Claims (i) and (iii) are proven in [26].

Regarding \( \subseteq \) in (ii): Let \( h \in \mathcal{P} \). Let \( L \in \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}(h) \) be a language and suppose \( \sigma \) is a locking sequence of \( h^* \) (as defined in Equation (2)) on \( L \). Let \( T \) be a text for \( L \). Let \( c_0 \) be such that \( \text{content}(\sigma) \subseteq \text{content}(T[c_0]) \). We show that \( T[c_0] \) is a locking sequence of \( h \) on \( L \), as desired. In particular, we show, for all \( \sigma' \supseteq T[c_0] \) such that \( \text{content}(\sigma') \subseteq L \), \( h^*(\sigma') = h^*(\sigma) \). Let \( \sigma' \supseteq T[c_0] \) such that \( \text{content}(\sigma') \subseteq L \).

Let \( \sigma'' \supseteq \sigma \) be such that \( \text{content}(\sigma'') = \text{content}(\sigma') \subseteq L \). As \( \sigma \) is stabilizing sequence of \( h^* \) on \( L \), \( h^*(\sigma'') = h^*(\sigma) \). Therefore, we have

\[
h^*(\sigma') = h(\text{content}(\sigma')) = h^*(\sigma'') = h^*(\sigma). \]

Thus, \( h \) is stabilizing on \( L \). \( \square \)

Note that [20] implies that \( [\mathcal{R} \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}] \subset [\mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}] \). This separation can also be shown using Theorem 3.6. The next theorem shows that, in both cases, we can get strong non-U-shapeness of the learner.

**Theorem 5.2.** We have that set-driven learning allows for strongly non-U-shaped learning, i.e.,

(i) \( [\mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}] = [\mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{N} \mathcal{U} \mathcal{E} \mathcal{x}] \) and

(ii) \( [\mathcal{R} \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}] = [\mathcal{R} \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{N} \mathcal{U} \mathcal{E} \mathcal{x}] \).

**Proof.** We will mostly prove both assertions at once. The inclusion \( \supseteq \) is trivial.

Regarding \( \subseteq \): Let \( h_0 \in \mathcal{P} \), let \( L = \mathcal{T} \mathcal{x} \mathcal{S} \mathcal{d} \mathcal{E} \mathcal{x}(h_0) \). For all finite sets \( D \), we let \( M(D) \) be the set of all \( D' \subseteq D \) such that

\[
\forall D' : D' \subseteq D' \subseteq D \Rightarrow h_0(D') = h_0(D').
\]

Let \( h_1 \in \mathcal{P} \) be such that

\[
\forall D : h_1(D) = \begin{cases} 
\uparrow, & \text{if } \exists D' \subseteq D : h_0(D') \uparrow; \\
\scriptsize{\text{pad}(h_0(D), \min(M(D))}, & \text{otherwise}. 
\end{cases} \tag{21}
\]

To help with Assertion (ii) of the theorem, note that

\[
h_0 \in \mathcal{R} \Rightarrow h_1 \in \mathcal{R}. \tag{22}
\]

**Claim 1.** \( h_1 \) is sink-stabilizing on \( L \).

**Proof of Claim 1.** Let \( L \in L \), let \( T \in \mathcal{T}(L) \). Using **Lemma 5.1**, \( h_0 \) stabilizes on \( L \); thus, let \( D_0 \subseteq L \) be the \( \leq \)-minimum such that

\[
(\forall D : D_0 \subseteq D \subseteq L) h_0(D) = h_0(D_0). \tag{23}
\]

Then we have
\[ \forall n : h_1(\text{content}(T[n])) = \text{pad}(h_0(D_0), D_0). \]  
(24)

Furthermore, for all \( D \) such that \( h_1(D) = \text{pad}(h_0(D_0), D_0) \) we have \( D \supseteq D_0 \). \( \square \) (FOR CLAIM 1)

The theorem now follows from Theorem 4.4. \( \square \)

With a similar proof as in Theorem 5.2, we can also show that rearrangement-independent learning allows for strongly non-U-shaped learning.

**Theorem 5.3.** We have that rearrangement-independent learners can be assumed total and strongly non-U-shaped, i.e.,

\[ [\mathcal{R}\text{TxtGEx}] = [\text{TxtEx}]. \]

**Proof.** The inclusion “\( \subseteq \)” is trivial.

“\( \supseteq \)” Let \( h_0 \in \mathcal{P} \), let \( \mathcal{L} = \text{TxtRiEx}(h_0) \). From [26] we know that, without loss of generality, \( h_0 \in \mathcal{R} \). Thanks to Lemma 5.1, we can suppose, without loss of generality, \( h_0 \) is stabilizing on \( \mathcal{L} \). For all \( D' \), \( c' \), \( D \), \( c \), we write \( \langle D', c' \rangle \rightarrow \langle D, c \rangle \) iff there is a text \( T \) such that \( \text{content}(T[c]) = D' \), \( \text{content}(T[c]) = D \) and \( c' \subseteq c \) (i.e., it is possible that at some point \( \langle D', c' \rangle \) is the input to a learner, and at some later point \( \langle D, c \rangle \) is the input).

Let \( \mathcal{R} \) be a computable predicate such that, for all \( D' \), \( c' \), \( D \), \( c \), \( R(D', c', D, c) \) iff \( \langle D', c' \rangle \rightarrow \langle D, c \rangle \) and

\[ \forall(D'', c''): (\langle D', c' \rangle \rightarrow \langle D'', c'' \rangle \rightarrow \langle D, c \rangle) \Rightarrow h_0(D'', c'') = h_0(D, c). \]

We define computable predicates \( f, h_1 \in \mathcal{P} \) such that

\[ \forall D, c : f(D, c) = \min\{\langle D', c' \rangle \mid \mathcal{R}(D', c', D, c)\}. \]

\[ \forall D, c : h_1(D, c) = \text{pad}(h_0(f(D, c)), f(D, c)). \]

Note that \( h_1 \in \mathcal{R} \). Further note that, for all \( x \), \( \varphi_{h_0(x)} = \varphi_{h_1(x)} \). Of course we have, as \( h_0 \) is stabilizing on \( \mathcal{L} \), for all \( L \in \mathcal{L} \) and all texts \( T \) for \( L \) that there is a \( c \) with

\[ (\forall D', c' \mid (\text{content}(T[c]), c) \rightarrow \langle D', c' \rangle \wedge D' \subseteq L) h_0(\text{content}(T[c]), c) = h_0(D', c'). \]

Hence, \( h_1 \) is stabilizing on \( \mathcal{L} \), and thus,

\[ \mathcal{L} \subseteq \text{TxtRiEx}(h_1). \]

**Claim 1.** \( h_1 \) is sink-stabilizing.

**Proof of Claim 1.** Let \( L \in \mathcal{L} \) and let \( T \) be a text for \( L \). Let \( D \subseteq L \) and \( c \) be such that \( h_1 \) stabilizes on \( \rho \subseteq T \) to \( \text{pad}(h_0(D, c), \langle D, c \rangle) \), i.e.,

\[ (\forall(D', c') \mid (\text{content}(\rho), \text{len}(\rho)) \rightarrow \langle D', c' \rangle ) h_1(D', c') = \text{pad}(h_0(D, c), \langle D, c \rangle). \]

(25)

It remains to show that \( \text{pad}(h_0(D, c), \langle D, c \rangle) \) is a sink for \( h_1 \) on \( L \), i.e., that for all \( \rho' \) with \( \text{content}(\rho') \subseteq L \) we have \( h_1^*(\rho \circ \rho') = \text{pad}(h_0(D, c), \langle D, c \rangle) \). Let \( \rho' \) be such that \( \text{content}(\rho') \subseteq L \). We have \( (D, c) \rightarrow (\text{content}(\rho), \text{len}(\rho)) \). Therefore, \( (D, c) \rightarrow (\text{content}(\rho \circ \rho'), \text{len}(\rho \circ \rho')) \), and thus, using (25), \( h_1^*(\rho \circ \rho') = \text{pad}(h_0(D, c), \langle D, c \rangle) \). \( \square \) (FOR CLAIM 1)

An application of Theorem 4.4 finishes the proof. \( \square \)

As \( [\text{TxtGEx}] = [\text{TxtRiEx}] \) \([37,25,26,28]\), we immediately get the following corollary, reproving a result from [3] and one from [21].

**Corollary 5.4.** We have that full-information learning allows for strongly non-U-shaped learning, and thus also for non-U-shaped learning, i.e.,

\[ [\text{TxtGNUEx}] = [\text{TxtGEx}] = [\text{TxtGSNUEx}]. \]

The following definition and the succeeding lemma are from [19]. These will be used for the proof of the final theorem for this paper, Theorem 5.7.

\[ ^{15} \text{Note that, for all } D \text{ and } c \text{ such that } \text{card}(D) \geq c, R(D, c, D, c). \]
Definition 5.5. For all $M \in \mathcal{P}$, $M$ is called canny iff, for all $\sigma$, (i) through (iii) below (recall from Section 2.1 that $M^*$ is the G-learner equivalent to the iterative learner $M$).

(i) $M^*(\sigma) \downarrow \Rightarrow M^*(\sigma) \in \mathbb{N}$, i.e., $M^*$ never outputs ?.
(ii) $M^*(\sigma \diamond \#) = M^*(\sigma)$.
(iii) For all $x \in \mathbb{N}$, if $M^*(\sigma \diamond x) \neq M^*(\sigma)$, then, for all $\tau \supseteq \sigma \diamond x$, $M^*(\tau \diamond x) = M^*(\tau)$.

Lemma 5.6. [Theorem 1 in [19]] For all $M' \in \mathcal{P}$, there is $M$ such that $\text{TxtItEx}(M') \subseteq \text{TxtItEx}(M)$ and $M$ is canny.

From the proof of [19, Theorem 1], it is easy to see that, if $M'$ is strongly non-U-shaped, then so is $M$. Note that the construction in that proof does not, in general, preserve non-U-shapedness.

From [19, Theorem 2] we have $[\text{TxtItEx}] = [\text{TxtItNUEx}]$. Contrasting this result, we have the following theorem.

Theorem 5.7. We have that there are non-U-shapedly iteratively learnable collections of languages which are not strongly non-shapedly learnable by an iterative learner, i.e.,

$$[\text{TxtItSNUEX}] \subset [\text{TxtItNUEx}].$$

Our proof makes use of padded ORT and, as noted above, for convenience, a self-learning class simpler to work with than that from Theorem 3.6.

Proof. It is easy to see that, in our present setting, the initial conjecture of any given iterative learner does not matter. Hence, for the following learner $M$, we will suppose the initial conjecture to be pad$(0, \emptyset)$. Let $M \in \mathcal{P}$ be such that

$$\forall e, D, x : M(\text{pad}(e, D), x) = \begin{cases} \text{pad}(e, D), & \text{if } x \in D \cup \{\#\}; \\
\uparrow, & \text{else if } \varphi_k(e) \uparrow; \\
\text{pad}(\varphi_k(e), D \cup \{x\}), & \text{else if } \varphi_k(e) \neq e; \\
\text{pad}(e, D), & \text{otherwise}. \end{cases}$$

Obviously, $M$ is canny. Let $\mathcal{L} = \text{TxtItEx}(M)$. From [19, Theorem 2] we have $\mathcal{L}$ is $\text{TxtItNUEx}$-learnable.

We proceed by showing that no canny $\text{TxtItEx}$-learner for $\mathcal{L}$ can be strongly non-U-shaped. By the remark after Lemma 5.6, any strongly non-U-shaped $\text{TxtItEx}$-learner can be assumed canny; hence, there is no $\text{TxtItSNUEX}$-learner for $\mathcal{L}$ if there is no canny $\text{TxtItSNUEX}$-learner for $\mathcal{L}$. Let $h \in \mathcal{P}$ be canny such that $\mathcal{L} \subseteq \text{TxtItEx}(h)$.

Let $p$ be a program for $h^*$ (where $h^*$ is again a starred learner as introduced in Section 2.1). We will make use of a Blum complexity measure $\Phi$ associated with $\varphi$; intuitively, for all programs $p$ and inputs $x$, $\varphi_p(x) \uparrow$ if $\Phi_p(x) \uparrow$, and the predicate $(p, x, t) \mapsto \Phi_p(x) \leq t$ is decidable (one can think of $\Phi_p(x)$ as the number of steps until program $p$ holds on input $x$). By the padded ORT, there are $a, b, e_1, e_2 \in \mathcal{R}$ strictly monotone increasing with pairwise disjoint ranges and $e_0 \in \mathbb{N}$ such that, abbreviating, for all $i, k$,

$$p_k(i) \iff h^*(a[k] \diamond b(k) \diamond a(i)) \downarrow = h^*(a[k] \diamond b(k)) \downarrow,$$

$$\overline{p}_k(i) \iff h^*(a[k] \diamond b(k) \diamond a(i)) \downarrow \neq h^*(a[k] \diamond b(k)) \downarrow,$$

$$E_k = \{a(i) \mid \exists j(\forall j' < j : p_k(j')) \land \Phi_p(a[k] \diamond b(k) \diamond a(j)) \geq i \}.$$ 

we have, for all $e, i, k$,

$$\varphi_{a(i)}(e) = \begin{cases} e_0, & \text{if } e \in \{?, e_0\}; \\
e_2(k), & \text{else if } e = e_1(k) \text{ and } i \geq k; \\
e, & \text{otherwise}; \end{cases}$$

$$\varphi_{b(k)}(e) = \begin{cases} e_1(k), & \text{if } e \in \{?, e_0\}; \\
e, & \text{otherwise}; \end{cases}$$

$$W_{e_0} = \text{content}(a);$$

$$W_{e_1(k)} = \text{content}(a[k]) \cup \{b(k)\} \cup \{E_k, \text{ if } \exists i : (\overline{p}_k(i) \land \forall j < i : p_k(j)); \emptyset, \text{ otherwise}; \}$$

$$W_{e_2(k)} = \text{content}(a[k]) \cup \{b(k)\} \cup E_k.$$
It is easy to see that \( W_{e_0} \in \mathcal{L} \) as witnessed by \( M \). Hence, \( h \) on the text \( a \) for \( W_{e_0} \) converges to a correct program number for \( W_{e_0} \). Let \( k \) be such that

\[
\forall k' \geq k : h^*(a[k']) = h^*(a[k])
\]

and

\[
W_{h^*(a[k])} = W_{e_0}.
\]

We now focus on the sets \( W_{e_1(k)} \) and \( W_{e_2(k)} \).

**Claim 1.** There is \( i \) such that \( \overline{P}_k(i) \) and \( \forall j < i : P_k(j) \).

**Proof of Claim 1.** Suppose, by way of contradiction, otherwise, i.e., suppose

\[
\neg[\exists i : (\overline{P}_k(i) \land \forall j < i : P_k(j))].
\]

This is equivalent to

\[
\forall i : [P_k(i) \lor h^*(a[i] \bowtie b(k) \bowtie a(i)) \downarrow \lor \exists j < i : \neg P_k(j)].
\]

(27)

Which is equivalent to

\[
[\forall i : P_k(i)] \lor [\exists i_0 : (h^*(a[i] \bowtie b(k) \bowtie a(i_0)) \downarrow \land \forall j < i_0 : P_k(j))].
\]

(28)

Note that (28) is implied by (27) by choosing the \( i_0 \) in the second disjunct of (28) to be the minimum \( i \) making the first disjunct of (27) false, if any.

**Subclaim:** \( E_k = \text{content}(a) \).

We proceed by considering possible cases of (28).

Suppose first that \( \forall i : P_k(i) \). Then, by the second disjunct in the definition of \( E_k \), \( E_k = \text{content}(a) \).

Suppose second that there is \( i_0 \) such that \( h^*(a[i] \bowtie b(k) \bowtie a(i_0)) \downarrow \) and, for all \( j < i_0 \), \( P_k(j) \). Then, by the first disjunct in the definition of \( E_k \), we get the subclaim. \( \square \) (FOR SUBCLAIM)

Using (26), we have that

\[
W_{e_1(k)} = \text{content}(a[k]) \cup \{b(k)\}.
\]

Further, by the Subclaim,

\[
W_{e_2(k)} = \text{content}(a[k]) \cup \{b(k)\} \cup \text{content}(a).
\]

It is easy to see that \( W_{e_1(k)} \) and \( W_{e_2(k)} \) are elements of \( \mathcal{L} \). Note that \( a[k] \bowtie b(k) \bowtie \lambda i.\# \) is a text for \( W_{e_1(k)} \), and, as \( h \) is canny, \( h \) converges on this text to \( h^*(a[k] \bowtie b(k)) \). Also note that \( a[k] \bowtie b(k) \bowtie a \) is a text for \( W_{e_2(k)} \), and, using (26) and the fact that \( h \) is an iterative learner, \( h \) on this text converges to \( h^*(a[k] \bowtie b(k)) \). This is a contradiction to \( W_{e_2(k)} \neq W_{e_1(k)} \), as \( h \) supposedly identifies both. \( \square \) (FOR CLAIM 1)

Let \( i_0 \) be the minimum \( i_0 \) as shown existent by Claim 1. Then \( E_k \) is finite, and, in particular, there is \( k' \) such that \( E_k = \text{content}(a[k']) \). Let \( k_1 = \max(k, k') \). Letting \( L = \text{content}(a[k_1]) \cup \{b(k)\} \), we have that

\[
W_{e_1(k)} = L = W_{e_2(k)}
\]

is a finite set. It is easy to see that \( L \in \mathcal{L} \). Note that

\[
a(i_0) \in L.
\]

(29)

As \( h \) is canny, we get, as \( a[k_1] \bowtie b(k) \bowtie \lambda i.\# \) is a text for \( L \),

\[
W_{h^*(a[k_1] \bowtie b(k))} = L.
\]

However, by choice of \( i_0 \),

\[
h^*(a[k_1] \bowtie b(k)) \neq h^*(a[k_1] \bowtie b(k) \bowtie a(i_0)).
\]

Using (29), we see that \( h \) is not strongly non-U-shaped. \( \square \)
6. Conclusion and open problems

Summing up, the following table indicates whether, for a given learning criterion (first column) semantic U-shapes (second column) or syntactic U-shapes (third column) can be avoided; in particular, “yes” in a column of the row of a learning criterion I means that every I-learnable set is I-learnable by a (strongly) non-U-shaped learner. Some of the justificatory remarks omit citing \( \text{TxtGEx} = [\text{TxtRIEx}] \) shown in [37,26].

<table>
<thead>
<tr>
<th>NU</th>
<th>SNU</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{TxtGEx}</td>
<td>yes, [3], Corollary 5.4</td>
</tr>
<tr>
<td>( \mathcal{R} \text{TxtRIEx} )</td>
<td>yes, Theorem 5.3</td>
</tr>
<tr>
<td>\text{TxtRIEx}</td>
<td>yes, Theorem 5.3</td>
</tr>
<tr>
<td>( \mathcal{R} \text{TxtRIEx} )</td>
<td>yes, Theorem 5.3</td>
</tr>
<tr>
<td>\text{TxtSdEx}</td>
<td>yes, Theorem 5.2</td>
</tr>
<tr>
<td>( \mathcal{R} \text{TxtSdEx} )</td>
<td>yes, Theorem 5.2</td>
</tr>
<tr>
<td>\text{TxtItEx}</td>
<td>yes, [19]</td>
</tr>
<tr>
<td>( \mathcal{R} \text{TxtItEx} )</td>
<td>open</td>
</tr>
</tbody>
</table>

We think that one can obtain \( [\mathcal{R} \text{TxtNUEX}] = [\mathcal{R} \text{TxtNUEX}] \) as a corollary to the proof in [19] of \( [\text{TxtNUEX}] = [\text{TxtNUEX}] \).

Trivially, we have

\[
[\text{TxtItCtrSNU} \subseteq [\text{TxtItCtrSU}] \subseteq [\text{TxtItCtrSZ}];
\]

\[
[\mathcal{R} \text{TxtItCtrSNU} \subseteq [\mathcal{R} \text{TxtItCtrSU}] \subseteq [\mathcal{R} \text{TxtItCtrSZ}].
\]

The other directions of inclusions are open.

As we can see from the overview, U-shaped learning is unnecessary in many different learning settings. Only very restricted settings in which learners make use of the hypothesis as a way to store information (as in the case of iterative learning), can sometimes require U-shapes.

Acknowledgments

We would like to thank the anonymous referees both of the conference version and the journal version of this paper for their comments and suggestions. Further, we want to thank Samuel E. Moelius III and Thomas Zeugmann for discussions on various theorems and their proofs. Major work on this paper was conducted while the second author was at the Department of Computer and Information Sciences, University of Delaware, USA.

References
