Avoiding Coding Tricks by Hyperrobust Learning

Matthias Ott[1] and Frank Stephan[2]

[1] Institut für Logik, Komplexität und Deduktionssysteme, Am Fasanengarten 5, Universität Karlsruhe, 76128 Karlsruhe, Germany, EU, m_ott@ira.uka.de

[2] Mathematisches Institut, Im Neuenheimer Feld 294, Universität Heidelberg, 69120 Heidelberg, Germany, EU, fstephan@math.uni-heidelberg.de

Abstract The present work introduces and justifies the notion of hyperrobust learning where one fixed learner has to learn all functions in a given class plus their images under primitive recursive operators. The following is shown: This notion of learnability does not change if the class of primitive recursive operators is replaced by a larger enumerable class of operators. A class is hyperrobustly Ex-learnable iff it is a subclass of a recursively enumerable family of total functions. So, the notion of hyperrobust learning overcomes a problem of the traditional definitions of robustness which either do not preserve learning by enumeration or still permit topological coding tricks for the learning criterion Ex. Hyperrobust BC-learning as well as the hyperrobust version of Ex-learning by teams are more powerful than hyperrobust Ex-learning. The notion of bounded totally reliable BC-learning is properly between hyperrobust Ex-learning and hyperrobust BC-learning. Furthermore, the bounded totally reliably BC-learnable classes are characterized in terms of infinite branches of certain enumerable families of bounded recursive trees. A class of infinite branches of a further family of trees separates hyperrobust BC-learning from totally reliable BC-learning.

1 Introduction

Self-reference and coding-tricks are an elegant way to prove many separation results in inductive inference. For example, the class of all functions f such that $f(0)$ is a program for f separates finite learning from the criterion Num which contains all classes that are subsets of enumerable families of total functions. Similarly, the class of all functions f where $f(0)$ is a program which computes f on almost all (but not necessarily all) places witnesses that BC-learning is more powerful than Ex-learning [5]. Such coding tricks allow to build simple proofs by using the following method: the less pretentious learner can evaluate the provided self-referential information — it is usually almost the desired output. But the more pretentious learner has to transform the information to information of higher quality (for example a program of a partial function into a program of a total extension) which turns out to be as difficult as well-known unsolvable recursion-theoretic problems.

Bārzdiņš proposed several notions of robust learning in order to find a concept of learning, where decoding self-referential information cannot be any longer the essential part of learning. In particular, he was interested into the question whether learning by enumeration is the only type of learning, where coding does not help. His basic hypothesis was that no kind of coding trick was preserved by all recursive operators. So he strengthened the notion of learning by requiring that not only the class S itself but also each image $\Theta(S)$ must be learnable for all suitable operators Θ. Clearly, Θ has to be recursive, but it was discussed whether Θ must map total functions to total ones, that is, must be general recursive. Jain, Smith and Wiehagen [8] analyzed this question and discovered that all proposed notions roughly behave like one of the following two cases: (a) The operator is only required to map the functions in S to total functions. Then robust learning does not even preserve Num: Some operator maps some class of constant functions onto the class of all recursive functions which is not learnable. (b) The operator is required to be general recursive, that is, the image of every total function has to be total. Then there is a class outside Num which is still robustly Ex-learnable.

Case (b) looks much more natural, since any notion of learning in the limit should cover Num, in particular, every class of constant functions should be learnable. In his original definition of robust learning, Fulk [7] followed this path and defined that

> a class S is *robustly learnable* if $\Theta(S)$ is learnable for all general recursive operators Θ.

Furthermore, he constructed already a class outside Num which is robustly learnable in the limit. Jain, Smith and Wiehagen [8] showed that Fulk's result can even be obtained using some topological kind of self-referential coding trick. They constructed a class of functions $f_1, f_2, \ldots$ which converge pointwise to one function f such that, for every general recursive operator Θ, either almost all $\Theta(f_k)$ are equal to $\Theta(f)$ and the class to be learned is finite or the point where $\Theta(f_k)$ and $\Theta(f)$ become different is, for almost all k, an upper bound on a program for f_k. Having such an upper bound, one can find a program for $\Theta(f_k)$ in the limit.

So, there is some demand to find a notion of robustness which on the one hand prevents the use of coding tricks and on the other hand preserves at least Num. The main idea to achieve this goal is to force the learner to cope with several images $\Theta(S)$ at the same time while keeping these operators restrictive enough to preserve at least learnability by enumeration. So, given any S,

> let $[S] = \{\Theta_e(f) : e = 0, 1, \ldots \text{ and } f \in S\}$ denote the closure of S under all *primitive recursive* operators $\Theta_0, \Theta_1, \ldots$ and define that S is *hyperrobustly* Ex-*learnable* iff $[S]$ is Ex-learnable.

Theorem 2.5 justifies this definition since it shows that the hyperrobustly learnable classes remain the same if one takes any larger enumerable class of operators instead of the one above. Furthermore, the notion of hyperrobust learning is compatible to standard robust learnability as used in [3, 8]: if S is hyperrobustly learnable then S is also robustly learnable. Moreover, if S is closed under finite variants then both notions are equivalent. The set $[S]$ is dense for every nonempty class S of functions, thus, hyperrobust learning cannot respect any bounds on the number of mind changes. Therefore, it is not suitable to look at mind change complexity in the context of hyperrobust learning and so, this paper focuses on the notions Num, Ex, BC and teams of Ex-learners or BC-learners.

This new notion of hyperrobust learning has also a further, more intuitive, motivation: Assume that a learner M can learn all axis-parallel rectangles in the plane. Certainly, one assumes that from M one can build a learner which additionally infers all rotated rectangles. However, clearly one does not want to build, for every different rotation Θ, a learner succeeding just on rectangles mapped by the rotation Θ. But instead one is interested in a learner which learns every image of any axis-parallel rectangle under *any* rotation Θ. The notion of hyperrobustness reflects this situation by requiring that *one* learning machine M learns every image of the functions in a class S under *all* primitive recursive operators.

For the reader's convenience, the definitions of Num, Ex and BC are included here: A learner M is a total recursive machine which receives as input initial segments σ of a total function f and outputs for every σ a guess for a program which is intended to represent a rule generating the function f. M learns f iff almost all guesses are programs computing f; M learns a whole class S of functions iff M learns every $f \in S$. The difference between the three criteria Num, Ex and BC is that a BC-learner need not satisfy any further requirements. But an Ex-learner has to converge explicitly, that is, for sufficiently large x the programs $M(f(0)f(1)\dots f(x))$ have to be the same. Num contains every class which is a subclass of an enumerable family of total recursive functions. One can infer the classes in Num by an easy algorithm called "learning by enumeration": the Ex-learner outputs always an index for the first function in the given family which is consistent with the data yet seen. "Num" stands for classes contained in a numbering. "Ex" stands for explanatory learning, that is, the learner converges to an explanation or program for f. "BC" stands for behaviourally correct learning, that is, the learner outputs almost always correct conjectures but the learner does not necessarily converge syntactically to one single program for f.

The notions of robustness can directly be transferred from Ex to BC: S is hyperrobustly BC-learnable iff $[S]$ is BC-learnable. For the ease of notation, if a result holds for explanatory learning as well as for behaviourally correct learning, then just the notion "learnable" is used in place of Ex-learnable and BC-learnable, respectively. If a result holds only for one of these two notions, then this notion is mentioned explicitly. Of course, when using the simplified notion "learnable", one has always to replace every occurrence of "learnable" consistently by either "Ex-learnable" or "BC-learnable", but one must not mix both notions.

The interested reader can find background information on recursion theory in the book of Odifreddi [11] and on inductive inference in the book of Osherson, Stob and Weinstein [12].

2 General Results

Hyperrobust learning differs from robust learning in the way that a hyperrobust learner succeeds on a class of functions S only if it succeeds on all images of S under primitive recursive operators. On the other hand, in the case of robust learning, it is only required that for every general recursive operator there is a learner succeeding with the data translated by this single operator.

Before defining robust and hyperrobust learning, the notions of operators are made more precise. An operator Θ is *general recursive* iff there is a program e such that, for every function f and every x, the program φ_e^f computes $\Theta(f)(x)$ by accessing f via builtin oracle calls and terminates. So, Θ maps every total function to a total function. Furthermore, the family $\Theta_0, \Theta_1, \ldots$ is *enumerable* iff there is a single general recursive operator Θ with $\Theta_e(f)(x) = \Theta(f)(e, x)$. On $\{0, 1\}$-valued functions, truth-table operators and general recursive operators mapping every total function to a total one coincide. In the general case, they are different as the example of the operator $\Theta(f)(x) = f(f(x))$ shows. Nevertheless, for the purposes of the present paper, it suffices to use the more restrictive variant and define primitive recursive operators as truth-table operators, that is, Θ is primitive recursive iff there are two primitive recursive functions g, h such that

$$\Theta(f)(x) = g(x, f(0)f(1)\ldots f(h(x)))$$

for all f and x.

Definition 2.1 Let S be a class of recursive functions.
(a) S is *robustly learnable* iff, for every general recursive operator Θ, there is a learner M which learns $\Theta(S) = \{\Theta(f) : f \in S\}$.
(b) S is *hyperrobustly learnable* iff there is one learner M which learns every function in $[S] = \{\Theta_e(f) : e = 0, 1, \ldots \text{ and } f \in S\}$ where $\Theta_0, \Theta_1, \ldots$ is an enumeration of all primitive recursive operators.

Note that in this general definition, one can define robust and hyperrobust learnability for all common notions of learning, like explanatory learning, behaviourally correct learning and so on. Hyperrobust learning satisfies two easy observations.

Fact 2.2 (a) *$[S]$ contains all primitive recursive functions since for every primitive recursive function g there is a primitive recursive operator Θ with $\Theta(f) = g$ for all functions f.*
(b) *No class S is hyperrobustly learnable with any bound on the number of mind changes, since $[S]$ is dense and dense classes cannot satisfy mind change bounds.*

These two facts establish a real difference to robust learning because there are classes of recursive functions which are even robustly learnable with at most one mind change [8]. On the other hand, for hyperrobust learning, the notions Ex, BC and their team-versions are the most interesting ones.

The definition of the mapping $S \to [S]$ and thus, also the definition of hyperrobustness is based on the class of primitive recursive operators. The decision to choose the class of primitive recursive operators may seem to be just arbitrary and one may wonder how other choices for the class of operators effect the notion of hyperrobustness. The next two results justify the definition: First it is shown that every hyperrobustly learnable class is bounded in the following sense.

Definition 2.3 A class S is bounded iff there is a total recursive function g which dominates every $f \in S$: $(\forall f \in S)\,(\exists x)\,(\forall y \geq x)\,[f(y) \leq g(y)]$.

Second it is shown that if, in the definition of hyperrobust learning, the enumeration $\Theta_0, \Theta_1, \ldots$ of all primitive recursive operators is replaced by a larger enumerable class of operators, then one still gets the same learning notion.

Theorem 2.4 *If S is hyperrobustly learnable then S is bounded.*

Proof It is sufficient to show the result for the more powerful notion of hyperrobust BC-learning. For function learning, it is convenient to represent the BC-learner as an NV$''$-learner M which predicts the function to be learned almost everywhere but which may be undefined at finitely many places as well as on any invalid data [15].

Now one defines inductively, for every f, the following function $\theta(f) = \lim_x \sigma_x$ starting with $\sigma_0 = \lambda$ and using $M_s(\sigma_x)$ as a notation for the result of $M(\sigma_x)$ after s computational steps (which is either undefined or $M(\sigma_x)$):

$$\sigma_{x+1} = \begin{cases} \sigma_x 1 & \text{if } M_s(\sigma_x)\!\downarrow = 0 \text{ for some } s \leq f(0) + f(1) + \ldots + f(x); \\ \sigma_x 0 & \text{otherwise.} \end{cases}$$

Since θ is a primitive recursive operator, M has to infer $\theta(f)$ for every $f \in S$. But whenever $\theta(f)(x)$ is 1, then M has made a prediction mistake and so, $\theta(f)$ takes only finitely often a value different from 0. Since M has to infer every primitive recursive function by Fact 2.2, M learns in particular all functions of the form $\sigma 0^\infty$. Thus the following function g is recursive:

$$g(x) = \max\{\min\{s : (\exists t < s)\,[M_s(\sigma 0^t)\!\downarrow = 0]\} : \sigma \in \{0,1\}^x\}.$$

Whenever $f(x) > g(x)$ then one finds within $f(x)$ stages some $t < f(x)$ such that $M(\sigma_x 0^t)\downarrow = 0$. So, the inductive definition of Θ diagonalizes for some $y \in \{x, x+1, \ldots, x+f(x)\}$ against the learner M, that is, $\sigma_{y+1} = \sigma_y 1$ while $M(\sigma_y) = 0$. Thus there exist at most finitely many x with $f(x) > g(x)$ and, therefore, g dominates f. Since the construction of g does not depend on the actual choice of f, g dominates every function in S. ▮

The next result shows that one does not change the notion of hyperrobust learning if one uses a more powerful enumerable family of general recursive operators instead of the family of all primitive recursive operators. As already mentioned above, this result provides an important justification of the model: the definition of hyperrobust learning does not depend on the actual choice of the class of operators as long as this class is "sufficiently rich" (for example, if the class contains all primitive recursive operators, or, all polynomial time computable operators). Clearly, if the class of operators contains only the identity operator, then hyperrobust and ordinary Ex-learning coincide and so, "sufficiently rich" is a necessary and natural postulate.

Theorem 2.5 *If S is hyperrobustly learnable then S is also hyperrobustly learnable with respect to any given enumerable family $\Theta_0, \Theta_1, \ldots$ of general recursive operators.*

Proof The main idea is the following: there is a function h with primitive recursive graph such that the operator Θ given by

$$f \to 0^{h(0)} f(0) 0^{h(1)} f(1) 0^{h(2)} f(2) \ldots$$

is primitive recursive and maps the function $\Theta_e(f)$ into $[S]$ for all $f \in S$ and every operator Θ_e. Then any hyperrobust learner M also infers every function $\Theta(\Theta_e(f))$ and can thus be translated into a learner succeeding on all functions $\Theta_e(f)$ by ignoring the zeros pasted into $\Theta_e(f)$ by Θ.

So, the main part of the proof is to show that h exists and that the concatenation $\Theta(\Theta_e)$ is primitive recursive for every operator Θ_e.

Given S, there exists a recursive function g which dominates all $f \in S$ by Theorem 2.4. Now, g is used in order to define the desired function h:

> $h(x)$ is the smallest number of computational steps s such that, for all $y \leq x$, all $e \leq x$ and all functions f with $f(z) \leq g(z)+x$, the computation $\Theta_e(f)(y)$ terminates within s steps.

The function h is well-defined since every operator Θ_e maps every total (not necessarily recursive) function onto a total function. The verification that h is recursive uses similar ideas like the proof of the folklore result that a Turing reduction which gives a total function for every oracle can be turned effectively into a truth-table reduction. Furthermore, one can primitive recursively check whether some computation halts within s stages if s is a lower bound for the input. So the graph of h is primitive recursive. To compute $\Theta(\Theta_e)(f)(x)$ one first checks whether x is of the form $h(0) + 1 + h(1) + 1 + \ldots + h(y)$. If so, one can compute $\Theta_e(f)(y)$ within $\max\{h(e), h(y)\}$ steps and output this value; in particular f is also only queried at places below $\max\{h(e), h(y)\}$. Otherwise, $\Theta(\Theta_e(f))(x) = 0$ and neither any computations nor any queries are necessary. So, $\Theta(\Theta_e)$ is a primitive recursive operator. ∎

Corollary 2.6 (a) *If S is hyperrobustly learnable then S is also robustly learnable in the sense that $\Theta(S)$ is learnable for every general recursive operator Θ.*
(b) *If S is closed under finite variants then S is hyperrobustly learnable iff S is robustly learnable.*

Proof Part (a) and the identical direction ($\Rightarrow$) of part (b) are a direct corollary of the preceding theorem. The direction ($\Leftarrow$) of part (b) is due to coding the index of Θ_e into the first argument of the function. Let $e' = \langle e, f(0) \rangle$ and define

$$\Theta(e' f(1) f(2) \ldots) = \Theta_e(f(0) f(1) f(2) \ldots)$$

where $\Theta_0, \Theta_1, \ldots$ is an enumeration of all primitive recursive operators. Then Θ is a general recursive operator. Furthermore, $[S] = \Theta(S)$, since, for every $f \in S$, the function $\Theta_e(f)$ is the image of the finite variant $e'f(1)f(2)\ldots$ which is also in S. So, $[S]$ has to be learnable and S is hyperrobustly learnable. ▮

Corollary 2.6 shows that hyperrobust learning is a natural generalization of robust learning: it is equivalent to first taking the closure under all finite variants and then applying a suitable general recursive operator Θ.

The notion of robustness was designed to destroy every numerical coding trick: for example, if $f(2x)$ is a program for f for almost all x, then the general recursive operator mapping f to $f(1)f(3)f(5)\ldots$ destroys this coding trick. Jain, Smith and Wiehagen [8] showed that topological coding — like, for example, coding the index into the branching points where the functions to be learned branch away from a fixed recursive function — cannot be destroyed by using a single general recursive operator. However, topological coding is destroyed by adding all finite variants of the functions in S to the class to be learned. Combining these two methods, that is, considering robust learning of classes closed under finite variants, one might hope that no coding tricks are left. Indeed, this hope is confirmed by the following characterization result which shows that the hyperrobustly Ex-learnable classes coincide with the classes in Num.

The criterion Num is quite prominent — Bārzdiņš, Leeuwen and Zeugmann [1, 17] showed that it coincides with further natural criteria: PEx-learning where the learner outputs only programs of total functions; NV-learning where the learner is total and predicts every $f \in S$ almost everywhere correctly; robustly totally reliable Ex-learning where a totally reliable Ex-learner either infers a function or diverges on it. Minicozzi [2, 10] introduced the notion of reliable learning; the difference between reliable Ex-learning and totally reliable Ex-learning is that in the second case the learner has also to diverge on nonrecursive functions while an ordinary reliable Ex-learner may behave arbitrarily on nonrecursive functions.

The next result adds hyperrobust learning to this list of characterizations of Num. So, every hyperrobust learnable class S can be learned by enumeration where the learner always outputs an index for the first recursive function, from a list of total recursive functions, which is consistent with the data seen so far.

Theorem 2.7 *A class S is hyperrobustly* Ex*-learnable iff it is in* Num.

Proof One direction is straightforward: If S is in Num, so is $[S]$. That is, if $S \subseteq \{f_0, f_1, \ldots\}$ for an enumeration $f_0, f_1, \ldots$ of total functions then $[S]$ is contained in the enumeration of all $g_{e,e'} = \Theta_{e'}(f_e)$ where $\Theta_0, \Theta_1, \ldots$ is an enumeration of all primitive recursive operators.

The converse direction is proven similar to Theorem 2.5. Assume that M Ex-learns $[S]$. Then M infers all functions of the form $\sigma 0^\infty$. Since M is an Ex-learner, one knows that, for every σ, either $\varphi_{M(\sigma)}(x)$ is defined for the first value $x \notin \mathrm{dom}(\sigma)$ or $M(\sigma 0^t) \neq M(\sigma)$ for almost all t. Let g be again a strictly increasing recursive function dominating every $f \in S$. Using g one defines inductively a function h with primitive recursive graph such that, for every

$\sigma \in \{0, 1, \ldots, g(x)\}^{h(0)+1+h(1)+1+\ldots+h(x-1)}$, either $\varphi_{M(\sigma),h(x)}(h(0)+1+h(1)+1+\ldots+h(x-1))$ has converged or $M(\sigma y 0^{h(x)}) \neq M(\sigma)$ for all $y \leq g(x)$. The function h is total since M has to infer every function which is almost everywhere 0. Now one defines again a primitive recursive operator Θ by

$$f \to 0^{h(0)} f(0) 0^{h(1)} f(1) 0^{h(2)} f(2) \ldots$$

and uses Θ to get an upper bound on the computation time of every $f \in S$: The learner M learns $\Theta(f)$ and its guesses $M(0^{h(0)} f(0) 0^{h(1)} f(1) 0^{h(2)} f(2) \ldots 0^{h(x)})$ are, for almost all x, equal to some value e. From the definition of h it follows that the computation $\varphi_e(h(0) + 1 + h(1) + 1 + h(2) + 1 + \ldots + h(x))$ converges within $h(x+1)$ steps to $f(x)$ for almost all x. Thus the function $x \to h(x+1)$ dominates the computation time for all $f \in S$, which implies that the class S is in Num. ∎

3 Hyperrobust BC-Learning is not Trivial

Within this section, it is shown that hyperrobust BC-learning does not collapse to Num as hyperrobust Ex-learning and attempts are made to characterize hyperrobust BC. A major tool in this research is the use of recursively bounded recursive trees [11, page 509], just called bounded recursive trees from now on. These trees are a generalization of binary recursive trees: for a bounded recursive tree T one can compute for every $\sigma \in T$ a complete list of the immediate successors in T which is impossible in the general case, even if σ has only finitely many successors. But it is still true when a recursive function bounds the size of the successors, that is, whenever $\sigma a \in T$ then $a \leq b(|\sigma|)$ for some fixed recursive function b. So, one can define a bounded recursive tree as a recursive function c which associates with every $\sigma \in T$ a finite and explicit list of all nodes $\sigma a \in T$. If c is primitive recursive then T is called a bounded primitive recursive tree.

A learning machine M is said to be *reliable* if M either converges to a correct program for the input function, or outputs infinitely often a signal for divergence, which, in the case of Ex-learning, can just be a mind change. Producing semantic mind changes alone is not sufficient to get a reliable version of BC-learning that differs from ordinary BC, as the following fact shows. This fact is based on two observations: First, behavioural correct learners can be made consistent. That is, the new consistent learner outputs for every input a hypothesis which is correct on the data seen so far [1, 6]. Second, consistent learners either converge semantically to the desired function or make infinitely many semantic mind changes.

Fact 3.1 *For every* BC-*learnable class* S *there is a* BC-*learner which either converges semantically or makes infinitely many semantic mind changes.*

Proof A given BC-learner M for S can be easily transformed into a new BC-learner N such that

$$\varphi_{N(\sigma)}(x) = \begin{cases} \sigma(x) & \text{if } x \in \mathrm{dom}(\sigma); \\ \varphi_{M(\sigma)}(x) & \text{otherwise.} \end{cases}$$

If M learns a function f, so does N since N changes the guess of M only on already known arguments by using the given values. If N semantically converges on f and almost always outputs some program of a fixed function ψ then $\psi = f$: for every x, there is a $\sigma \preceq f$ such that $N(\sigma)$ computes ψ and $x \in \mathrm{dom}(\sigma)$. It follows that $\psi(x)\downarrow = \sigma(x) = f(x)$. So, N learns a function f iff N converges semantically on f. ▮

Looking a bit closer, it even holds that N learns a function f iff N outputs infinitely often the same program during the inference of f. Therefore, the analogue of reliable learning for BC must signal divergence more explicitly. A suitable definition is the following: The reliable BC-learner indicates divergence either by outputting a special value like "?" or by making a definitely wrong prediction where the underlying BC-learner is given by an NV$''$-prediction machine M [5, 15], which, by definition, is successful on f if

$$(\forall f \in S)\,(\forall^\infty x)\,[M(f(0)f(1)\ldots f(x))\downarrow = f(x+1)].$$

A more restrictive variant is *totally reliable learning* [2, 10] where the learner has to signal divergence not only on all recursive functions not learned but also on all nonrecursive function which cannot be learned by definition.

Definition 3.2 M is a reliable BC-learner iff, for every recursive function f, either M BC-learns f by predicting almost always the correct value (that is, for almost all x, $M(f(0)f(1)\ldots f(x))\downarrow = f(x+1)$) or M diverges on f by infinitely often outputting either ? or a defined but wrong prediction. M is a totally reliable BC-learner if M diverges also on every nonrecursive function.

Zeugmann [17] observed that robustly totally reliably Ex-learnable classes are just those in Num. A related result is that, for bounded classes, Num is equal to totally reliable Ex. Together with Theorem 2.7, one obtains the following equivalence.

Fact 3.3 *For a bounded class S the following statements are equivalent.*
(a) *S is in* Num.
(b) *S is totally reliably* Ex*-learnable.*
(c) *S is hyperrobustly* Ex*-learnable.*

The central question of this section is to what extent the equivalence above transfers to BC. The next characterization of bounded totally reliably BC-learnable classes is an important tool to attack this question.

Theorem 3.4 *A bounded class S is totally reliably* BC*-learnable iff there is a family $T_0, T_1, \ldots$ of bounded recursive trees such that every tree has only finitely many infinite branches and every $f \in S$ is the infinite branch of such a tree.*

Proof ($\Rightarrow$): Assume that g bounds S and M is an NV$''$-predictor for S which in addition signals infinitely often divergence on every function f which M does not learn. Now let the tree T_σ contain all prefixes of σ plus all $\eta \succeq \sigma$ such that

- $\eta(x) \leq g(x)$ for all $x \in \mathrm{dom}(\eta) - \mathrm{dom}(\sigma)$ and
- there are no τ, a with $\sigma \preceq \tau a \prec \eta$ and $M_{|\eta|}(\tau)\downarrow \neq a$

where, of course, the special symbol "?" is different from a.

Clearly, the T_σ form a recursive family of trees bounded by g. Assume now that T_σ has infinitely many infinite branches. As a consequence of König's Lemma and the fact that T_σ is finitely branching, there is an infinite branch f which is not isolated. This implies that M cannot predict f at almost all points correctly. So, on input f, divergence must also be signaled above σ by M, which contradicts the fact that f is an infinite branch of T_σ.

For every $f \in S$, there is a prefix $\sigma \preceq f$ such that M predicts f correctly after seeing σ and all x with $f(x) > g(x)$ are in $\mathrm{dom}(\sigma)$. Then it follows from the definition that f is an infinite branch of T_σ and direction ($\Rightarrow$) is completed.

($\Leftarrow$): Let $T_0, T_1, \ldots$ be a family of bounded recursive trees such that every tree has only finitely many infinite branches and every function in S is branch of such a tree. Without loss of generality, the family is dense in the sense that for every σ there is a tree containing σ. This can be achieved by adding all finite trees of the form $\{\tau : \tau \preceq \sigma\}$ to the list. The new family is still enumerable and the class of functions on trees in the family remains the same. Let $T[\tau]$ denote all nodes of the tree T which are comparable to τ. Now the totally reliable BC-learner works as follows:

> $M(\sigma)$ finds the first tree T_e with $\sigma \in T_e$.
> If there was a recent change of the tree, that is, if there is $e' < e$ with $\tau \in T_{e'}$ for all $\tau \prec \sigma$ then $M(\sigma) = ?$ in order to signal divergence.
> Otherwise $M(\sigma)$ searches for an a such that $T_e[\sigma b]$ is finite for all $b \neq a$ and $M(\sigma) = a$ if such an a is found.

The first step of the algorithm is well-defined since every σ is node of some tree T_e.

If f is not infinite branch of any tree T_e then, during the inference of f, M signals infinitely often divergence, since M has infinitely often to change the tree.

If f is infinite branch of some tree then there is a first such tree T_e in the enumeration. For sufficiently large $\sigma = f(0)f(1)\ldots f(x)$, f is the only infinite branch of $T_e[\sigma]$ and $\sigma \notin T_{e'}$ for any $e' < e$. Now $f(x+1)$ is the unique value a with $T_e[\sigma a]$ being infinite. Since the trees T_e are uniformly bounded recursive, a suitable search algorithm finds the value $f(x+1)$. Therefore, M predicts f almost everywhere, that is, $M(f(0)f(1)\ldots f(x))\downarrow = f(x+1)$ for almost all x.

So, for every function f, M either BC-learns f (in the prediction model) or M signals infinitely often divergence. ∎

The next theorem establishes some compatibility between the various notions of reliable learning. It shows that reliable Ex-learning and totally reliable BC-learning are generalizations of totally reliable Ex-learning in two disjoint directions.

Theorem 3.5 *A class S is totally reliably* Ex*-learnable iff S is reliably* Ex*-learnable and totally reliably* BC*-learnable.*

Proof The direction ($\Rightarrow$) is clear, for the converse direction ($\Leftarrow$) note that one can count in the limit the number of signals for divergence. So, there is a recursive function H such that H converges on f to c, if the totally reliable BC-learner, on input f, signals exactly c times divergence, and such that H converges on f to ∞, if the totally reliable BC-learner signals infinitely often divergence. Let M be a reliable Ex-learner and let *pad* be an injective padding function. Now the new learner N is given by

$$N(\sigma) = pad(M(\sigma), H(\sigma)).$$

If $f \in S$ then N converges to $pad(e, c)$ where e is the program to which M converges on f and c is the finite number of (false) signals for divergence produced the totally reliable BC-learner for S on f. If N converges on f, then H converges on f to a finite number and thus f is recursive. Furthermore, also M must converge to a program e and since M is reliable and f is recursive, $pad(e, c)$ is a program for f. Thus N is totally reliable. ∎

Case, Kaufmann, Kinber and Kummer [4] showed that there is a family of binary recursive trees of width 2 whose infinite branches are not Ex-learnable. This yields a class S which is totally reliably BC-learnable but not reliably Ex-learnable. For bounded classes, the concept of totally reliable BC-learning is also a proper generalization of Num. Together with the next result one obtains that the three notions from Fact 3.3 become all different for BC: Num is properly included in bounded totally reliable BC, which is properly included in hyperrobust BC.

Theorem 3.6 *If S is a bounded and totally reliably* BC*-learnable class then S is also hyperrobustly* BC*-learnable. But the converse implication does not hold.*

Proof For the first statement, let S be bounded and totally reliably BC-learnable. There is a uniformly recursive family $T_0, T_1, \ldots$ of trees such that every tree has only finitely many infinite branches and every function in S is infinite branch of some tree T_e. Without loss of generality, one can assume that for every e and every primitive recursive operator Θ there is some e' such that $T_{e'} = \Theta(T_e)$. Otherwise, $T_0, T_1, \ldots$ can be replaced by another uniformly recursive family of trees which contains the trees T_i and which is closed under all primitive recursive operators. So, whenever $f \in S$ and Θ is a primitive recursive operator then there is some tree T_e such that f is infinite branch of T_e and also of some further tree $T_{e'} = \Theta(T_e)$. This implies that $\Theta(f)$ is an infinite branch of $T_{e'}$. So, the class S' of all infinite branches of $T_0, T_1, \ldots$ contains $[S]$. By Theorem 3.4 there is a totally reliable BC-learner M for S'. Now M is also a BC-learner for $[S]$ and, thus, already a hyperrobust BC-learner for S.

The second statement can be proven using the following idea: one constructs a family of binary trees $T_0, T_1, \ldots$ such that, for every tree T_e and every primitive recursive operator Θ_i, the infinite branches of the image tree $\Theta_i(T_e)$ are either isolated or nonrecursive. Furthermore, T_e diagonalizes against the learner M_e from an enumerable list $M_0, M_1, \ldots$ of all learners in such a way that whenever M_e is a totally reliable BC-learner then T_e has only one infinite branch on which M_e infinitely often signals divergence. The class

$$S = \{\Theta_i(f) : \Theta_i(f) \text{ is recursive and } f \text{ is on } T_e \text{ for some } i, e\}$$

witnesses the separation. The details of the construction of the T_e and the verification that S is hyperrobustly BC-learnable but not totally reliable BC-learnable are omitted due to space constraints. ▮

Zeugmann [17] showed that a totally reliably Ex-learnable class S is also robustly learnable under this criterion iff S is in Num. One can deduce from it that S is robustly totally reliably Ex-learnable iff S is bounded. The same holds for robust totally reliably BC-learnability based on Zeugmann's diagonalization strategy for unbounded classes S and on an adaptation of the implication "bounded totally reliable BC $\Rightarrow$ hyperrobust BC" for bounded classes S.

Theorem 3.7 *A totally reliably* BC*-learnable class S is also robustly learnable under this criterion iff S is bounded.*

4 Team-Learning and the Union-Theorem

Bārzdiņš [1, 2] showed that there are explanatory learnable classes S_1, S_2 such that their union is not explanatorily learnable. This result easily generalizes to the fact that there are unions of $n+1$ learnable classes which are not contained in the union of n learnable classes [16]. Pitt and Smith [13, 14] showed, that these unions can also be characterized in terms of probabilistic learners and teams of learners, that is, the following statements are equivalent for Ex-learning as well as for BC-learning.

- S is contained in the union of n learnable classes.
- Some probablistic machine learns S with some probability p where $p > \frac{1}{n+1}$.
- A (h,k)-team with $\frac{h}{k} > \frac{1}{n+1}$ learns S in the sense that there are k learners such that, for every $f \in S$, at least h of them learn f.

Note that the probability and the fraction $\frac{h}{k}$ of successful machines in the team have to be really greater than $\frac{1}{n+1}$ and cannot be equal to this value, since a team of $k = h \cdot (n+1)$ learners, where h learners have to succeed, can already infer the union of $n+1$ learnable classes: the first h learners follow the algorithm to learn S_1, the second h learners follow the algorithm to learn S_2, ..., the last h learners follow the algorithm to learn S_{n+1}.

For hyperrobust Ex-learning, one can show that this connection between team-learning on the one side and unions on the other side does no longer hold. The hyperrobustly Ex-learnable classes are closed under union but teams of $n+1$ hyperrobust Ex-learners are more powerful than teams of n learners. An intuitive explanation for this fact is that if $[S_1 \cup S_2]$ needs a team of two Ex-learners then so does $[S_1]$ or $[S_2]$. So, the closure operation does not permit to split a class of functions into two classes which are really easier to learn.

Fact 4.1 *If S_1 and S_2 are hyperrobustly* Ex*-learnable, so is $S_1 \cup S_2$.*

The result follows from the equivalence of hyperrobust Ex-learning and Num and from the fact that Num is closed under union. The next result establishes that the team hierarchies for hyperrobust Ex-learning and hyperrobust BC-learning are proper.

Theorem 4.2 *The team hierarchy for hyperrobust learning is proper.*

Proof Let S_k be the set of all functions which are infinite branch of some bounded primitive recursive tree of rank up to k.

Given f, the learning algorithm first finds (in the limit) a tree T such that f is an infinite branch of T. Having found this tree T, one uses the algorithm of Case, Kaufmann, Kinber and Kummer [4] who showed that knowing an index of the tree and having a primitive recursive function majorizing all infinite branches, one can learn the function by a team of $k+1$ Ex-learners or k BC-learners, respectively. The team-size is also optimal. The class S_k is closed, that is, $[S_k] = S_k$. So, it follows that S_k is learnable by a team of hyperrobust learners of size k (BC) and $k + 1$ (Ex), respectively, but not by a smaller team. ∎

Furthermore, for hyperrobust Ex-learning, one can even show that there exists a proper team hierarchy *within* the class of all hyperrobustly BC-learnable functions. The n-th level of this hierarchy is given by the class of all infinite branches of bounded primitive recursive trees of width up to n, that is, of trees which have in every depth at most n nodes.

5 Conclusion

The research on robust learning has the goal to investigate whether there are learning notions which make it impossible to learn a function by evaluating self-referential coding-information in the graph of the function. The previous approaches to consider all classes $\Theta(S)$ either still allowed some topological kind of coding [7, 8] or permitted partial operators which already destruct the basic algorithm "learning by enumeration". The authors believe that such a basic algorithm should be preserved and therefore propose a new approach: the learner has to deal with all images of general recursive operators $\Theta(S)$ simultaneously. The collection of operators used must nevertheless be restricted since permitting all operators would mean to postulate the learning of all recursive functions. It is shown that using all primitive recursive operators is a reasonable choice. In particular, the following two results justify this notion: first, all sufficiently powerful families of operators give the same notion of learning; second, a class S is hyperrobustly learnable with respect to this choice of operators iff the closure of S under finite variants is robustly learnable with respect to the traditional definition. Hyperrobust Ex-learning meets Bārzdiņš' hypothesis since it collapses to Num. But the hyperrobust versions of BC-learning and team-learning permit the inference of classes outside Num. There are relations between hyperrobust BC-learning and totally reliable BC-learning. Furthermore, families of bounded recursive trees turn out to be a useful tool for investigating hyperrobust learning and for characterizing totally reliable BC-learning of bounded classes.

Acknowledgment Frank Stephan is supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-2.

References

1. Janis Bārzdiņš. Two theorems on the limiting synthesis of functions. *In Theory of Algorithms and Programs, Latvian State University, Riga*, 210:82–88, 1974.
2. Leonard Blum and Manuel Blum. Towards a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
3. John Case, Sanjay Jain, Matthias Ott, Arun Sharma and Frank Stephan. Robust learning aided by context. In *Proceedings of Eleventh Annual Conference on Computational Learning Theory* (COLT), pages 44–55, ACM Press, New York, 1998.
4. John Case, Susanne Kaufmann, Efim Kinber and Martin Kummer. Learning recursive functions from approximations. *Journal of Computer and System Sciences*, 55:183–196, 1997.
5. John Case and Carl Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.
6. Jerome Feldmann. Some decidability results on grammatical inference and complexity. *Information and Control*, 20:244–262, 1972.
7. Mark Fulk. Robust separations in inductive inference. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science* (FOCS), pages 405–410, St. Louis, Missouri, 1990.
8. Sanjay Jain, Carl Smith and Rolf Wiehagen. On the power of learning robustly. In *Proceedings of Eleventh Annual Conference on Computational Learning Theory* (COLT), pages 187–197, ACM Press, New York, 1998.
9. Wolfgang Merkle and Frank Stephan. Trees and learning. *Proceedings of the Ninth Annual Conference on Computational Learning Theory* (COLT), pages 270–279, ACM Press, New York, 1996.
10. Eliana Minicozzi. Some natural properties of strong-identification in inductive inference. *Theoretical Computer Science*, 2:345–360, 1976.
11. Piergiorgio Odifreddi. *Classical Recursion Theory.* North-Holland, Amsterdam, 1989.
12. Daniel Osherson, Michael Stob and Scott Weinstein. *Systems that Learn.* MIT Press, Cambridge, Massachusetts, 1986.
13. Lenny Pitt. Probablistic inductive inference. *Journal of the Association of Computing Machinery*, 36:383–433, 1989.
14. Lenny Pitt and Carl Smith. Probability and plurality for aggregations of learning machines. *Information and Computation*, 77:77–92, 1998.
15. Karlis Podnieks. Comparing various concepts of function prediction, Part 1. *Theory of Algorithms and Programs, Latvian State University, Riga*, 210:68–81, 1974.
16. Carl Smith. The power of pluralism for automatic program synthesis *Journal of the Association of Computing Machinery*, 29:1144–1165, 1982.
17. Thomas Zeugmann. On Bārzdiņš' conjecture. In K. P. Jantke, editor, *Proceedings of the International Workshop on Analogical and Inductive Inference* (AII'86), volume 265 of LNCS, pages 220–227. Springer, 1986.