On Learnability with Computable Learners

Sushant Agarwal SUSHANT.AGARWAL@UWATERLOO.CA
Nivasini Ananthakrishnan NANANTHA@UWATERLOO.CA
Shai Ben-David SHAI@UWATERLOO.CA
Tosca Lechner TLECHNER@UWATERLOO.CA
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada

Ruth Urner RUTH@EECS.YORKU.CA
Lassonde School of Engineering, EECS Department
York University, Toronto, ON, Canada

Editors: Aryeh Kontorovich and Gergely Neu

Abstract

We initiate a study of learning with computable learners and computable output predictors. Recent results in statistical learning theory have shown that there are basic learning problems whose learnability can not be determined within ZFC (Ben-David et al. (2017, 2019)). This motivates us to consider learnability by algorithms with computable output predictors (both learners and predictors are then representable as finite objects). We thus propose the notion of *CPAC learnability*, by adding some basic computability requirements into a PAC learning framework. As a first step towards a characterization, we show that in this framework learnability of a binary hypothesis class is not implied by finiteness of its VC-dimension anymore. We also present some situations where we are guaranteed to have a computable learner.

Keywords: Computability, PAC learning, VC-Dimension

1. Introduction

A recent study came up with a rather surprising result: there are basic learning problems whose learnability (even in the sense of *weak learning*) cannot be determined by the common notions of mathematical proofs, the set theory ZFC (Ben-David et al. (2017, 2019)). One naturally wonders what could be the reason for such unprovabilty.

A closer look reveals that the notion of learnability refers to the existence of *learners*, which are mappings from training samples to, say, classifiers (or other possible learned objects). The common statistical learning theory, in which we have the fundamental characterization of PAC learnability by the finiteness of the VC-dimension, allows for the learners to be arbitrary functions. Had we required learners to be computable, there would have been a finite representation for each learner (as the code for the program implementing it), ruling out independence of ZFC results of the type shown in Ben-David et al. (2017, 2019). Another merit of strengthening the definitions of learnability by allowing only computable learners is that it would better reflect our intention of modeling automated learning.

In this paper we wish to initiate an investigation of the nature of such notions of computable learning. We focus our attention on binary classification learning. The purely statistical Vapnik-Chervonenkis theory provided a characterization of uniform convergence by a combinatorial prop-

erty of the hypothesis class (the VC-dimension) for which learners were considered as general functions. VC-theory did not impose any requirement on the learners actually being implementable by algorithms (Vapnik and Chervonenkis, 1971). Valiant's computational learning theory framework of PAC learnability combined the statistical success condition with a requirement that learners are algorithms whose running time is polynomial in $\frac{1}{\epsilon}, \log\left(\frac{1}{\delta}\right)$ and some parameter d of the function class, for example, the Euclidean dimension of the feature space in the case of linear classifiers (Valiant, 1984; Haussler, 1992). In this work, we aim at introducing an intermediate setup. While we do not impose any (polynomial) efficiency requirements on the the runtime of our learners, we will require the learners to be computable. We then demonstrate settings in which such a definition leads to different conclusions than what we know for the common notion of PAC learning.

In Section 2.2, we develop our modified definition of PAC learnability, which incorporates a requirement of the learning algorithm and its outputs to be computable. We term this new notion CPAC learnability. Of course, restricting the set of candidate learners can only result in a decrease in the scope of learnable classes. Therefore, the first step we take is asking: are there hypotheses classes of finite VC-dimension (therefore, PAC learnable classes) that are not CPAC learnable? We then move on to analyze and identify conditions under which such classes exist. We also consider various variations of the basic setup: proper versus non-proper learning, as well as a notion of non-uniform learnability.

The main contribution of this work is to show the existence of such classes (that are learnable under the common, unrestricted definition of PAC learning, but cannot be learned by any computable learner) in the setting of *proper learning*, that is, under the additional requirement that the learner output a function from the hypothesis class; the existence of such classes shows that the characterization of PAC learnability by the VC-dimension (Vapnik and Chervonenkis, 1971; Vapnik, 2000) crucially depends on disregarding computability requirements. These results are presented in Section 4. In that section, we also identify conditions under which usual PAC learnability does imply CPAC learnability, that is, under which computable learners exist. We also develop some insights into the implications of computability requirements in the more challenging (to analyze) case of *improper learning* in Section 6. Furthermore, in Section 5 we extend our study to a notion of *non-uniform learnability* and show that in this setting there are also classes that are learnable (namely the class of all computable functions), but not learnable by a computable learner.

2. Setup

2.1. General background

Computability Let $\Sigma = \{0, 1\}$ be a binary alphabet and let Σ^* be the set of all finite words over Σ. Note that we can naturally identify Σ^* with the natural numbers $\mathbb{N}$ or with the set of all finite subsets of natural numbers. We will often implicitly assume that we fixed one such encoding.

We further assume that we fix some programming language and thus use the existence of Turing machines synonymously with the existence of some *program* or *algorithm* (in our fixed language). A function $f : \Sigma^* \to \Sigma^*$ is said to be *computable* if there exists a program P that halts on every input $\sigma \in \Sigma^*$ and we have $P(\sigma) = f(\sigma)$ for every $\sigma \in \Sigma^*$. A subset S of Σ^* is called *recursively enumerable (RE)* if there exists a program P that takes natural numbers as input, halts on every input and whose range is S. We call a set $S \subseteq \Sigma^*$ *decidable* if there exists a program P that halts on every input $\sigma \in \Sigma^*$ and outputs 1 if $\sigma \in S$ and outputs 0 otherwise.

Learning We now first recall the standard learning theoretic framework. We let $X = \mathbb{N}$ denote the domain and $Y = \{0, 1\}$ denote the label space. A *hypothesis* is a function $h : X \to Y$. We will often identify such binary functions h with the subset of the domain that h maps to 1 and denote this as $X_h = h^{-1}(1)$. A *hypothesis class* $\mathcal{H} \subseteq Y^X$ is a set of hypotheses.

As is common in learning theory, we assume that data is generated by some distribution D over $X \times \{0, 1\}$. We denote the *error* of a hypothesis h with respect to the distribution D by $L_D(h) = \text{Prob}_{(x,y)\sim D}[h(x) \neq y]$. A *learner* is a function that takes in a finite sequence of labeled domain points $S = ((x_1, y_1), \ldots, (x_n, y_n))$ and outputs a hypothesis h. The *empirical error* of a hypothesis h with respect to a sample $S = ((x_1, y_1), \ldots, (x_n, y_n))$ is defined as $L_S(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[h(x) \neq y]$

Definition 1 ((Agnostic) PAC learnability) *A hypothesis class $\mathcal{H}$ is agnostic PAC learnable if there exists a learner A such that for all $\epsilon, \delta \in (0, 1)$, there is a sample size $m(\epsilon, \delta)$ such that, for any distribution D, if the input to A is an iid sample S from D of size at least $m(\epsilon, \delta)$, then, with probability at least $(1 - \delta)$ over the samples, the learner outputs a hypothesis $h = A(S)$ with*

$$L_D(h) \leq \inf_{h \in \mathcal{H}} L_D(h) + \epsilon$$

The class is said to be PAC learnable in the realizable case*, if the above holds under the condition that* $\inf_{h\in\mathcal{H}} L_D(h) = 0$.

Definition 2 (Proper learning) *A hypothesis class $\mathcal{H}$ is* proper (agnostic) PAC learnable *if it is (agnostic) PAC learnable with a learner that always outputs a hypothesis $h \in \mathcal{H}$ from the class.*

The standard notion of learnability (as in Definition 1) reflects a guarantee that holds uniformly for all functions in the hypothesis class $\mathcal{H}$ (and uniformly overall data-generating distributions). Namely, from a certain sample size on, the output of the learner has to compete with the *best* hypothesis in $\mathcal{H}$. We now introduce a notion of *non-uniform learnability* where we allow the sample size that the learner requires to depend on the hypothesis $h \in \mathcal{H}$ that it is aiming to compete with.

Definition 3 (Non-uniform learnability) *We say that a hypothesis class $\mathcal{H}$ is* non-uniformly learnable *if there exists a learner A and a function $m : (0, 1)^2 \times \mathcal{H} \to \{0, 1\}$ such that, for every $\epsilon, \delta \in (0, 1)$, every hypothesis $h \in \mathcal{H}$ and for every distribution D, if $m \geq m(\epsilon, \delta, h)$, then with probability of at least $1 - \delta$ over the choice of $S \sim D^m$, it holds that*

$$L_D(A(S)) \leq L_D(h) + \epsilon.$$

2.2. CPAC learnability

Computable learning We now develop our framework of CPAC learnability. While we do not impose any (polynomial) efficiency requirements on the the runtime of our learners, we will require the learners to be computable. In addition we would like to also impose computability requirements on the output hypotheses of these learners and for the hypothesis class $\mathcal{H}$. That is, we would like to ensure that a claim of "learnability" for a hypothesis class implies the existence of an algorithm that implements the learning process and that this learning algorithm provides an output hypothesis in a form that allows for (computably) evaluating this hypothesis on all inputs of the domain X.

Remark 4 *It is not straightforward to define what a "computable class of functions" is. Computability is defined as a property of sets of finite words. Here we are considering sets of infinite objects – functions from* $\mathbb{N}$ *to* $\{0,1\}$*. Of course, we may wish to consider only computable functions, and in that case each function can be represented by a finite program. However, while a* set of programs *may be decidable, this is not naturally implied by "simplicity" of the* set of functions *these programs encode. Note that even for a set containing only one simplest function, say the constant-zero function, the set of all programs that encode it (in any fixed given programming language) is not decidable.*

In light of the above discussion, we pose as a minimal requirement for our hypothesis classes that they consist of computable functions. Further, we introduce the following two additional restrictions on the computability properties (decidable and recursively enumerable) of the set of programs that compute these functions and that the learning algorithm will use to output a hypothesis. Thus, we distinguish the following two notions for the representations for hypothesis classes:

Definition 5 (Decidable Representation (DR) of a Hypothesis class) *We say that a class of functions,* $\mathcal{H}$*, is* Decidably Representable (DR) *if there exists a decidable set of programs* $\mathcal{P}$ *such that the set of all functions computed by a program in* $\mathcal{P}$ *equals* $\mathcal{H}$*.*

Definition 6 (Recursively Enumerable Representation (RER) of a Hypothesis class) *We say that a class of functions* $\mathcal{H}$ *is Recursively Enumerably Representable (RER) if there exists a recursively enumerable set of programs* $\mathcal{P}$ *such that the set of all functions computed by a program in* $\mathcal{P}$ *equals* $\mathcal{H}$*.*

Remark 7 *There are other ways of finitely representing functions over a countable domain. For example, we may restrict our attention to functions that have finite support, that is, functions that are constant for all large enough inputs:*

$$\mathcal{F} = \{f \in \{0,1\}^{\mathbb{N}} \mid \exists\, n_f \in \mathbb{N} \text{ such that } \forall\, i,j > n_f,\ f(i) = f(j)\}.$$

Functions in $\mathcal{F}$ *can be represented by explicitly listing the finite set of instances on which their value differs from the value that the function converges to. Most of the hypothesis classes we introduce in Section 4 consist of such functions with finite support, and we will by default assume the list representation. It is easy to see that such a list could be turned into a program that computes the function, thus all functions in* $\mathcal{F}$ *are computable. However, subsets of* $\mathcal{F}$ *may not be DR or RER in the sense of the above Definitions 5 and 6.*

We now introduce our notion of *computable PAC learnability*, or *CPAC learnability* for short. The minimal requirement on the output of a CPAC learner is that it uses a representation that allows for evaluating the output hypothesis on every input of the domain.

Definition 8 (CPAC learnability) *We say that a class* $\mathcal{H}$ *is* (agnostic) CPAC *learnable, if there is a computable (agnostic) PAC learner for* $\mathcal{H}$ *that uses a representation for the predictors it outputs, that allows it to evaluate the outputted function on each domain point. If the learner always outputs a (representation of) a hypothesis in class* $\mathcal{H}$*, we call it a* proper CPAC learner *and the class* proper CPAC learnable.

In addition, we may require that the CPAC learner uses representations according to the definitions of DR and RER classes.

Without explicitly defining these here, we note that the above also imply notions of non-uniform learnability with additional requirements for the learner to be a proper learner with respect to $\mathcal{H}$ and/or for the learner to be computable.

3. Summary of Results

We consider various learning scenarios and investigate whether learnability in the general sense (where learners are modeled as arbitrary functions) implies learnability with computable learners. We summarize our results in the tables below. As discussed earlier, we only consider classes of computable predictors, thus "any class" assumes only this. Note also that a class being Decidably Representable (DR) implies that it is Recursively Enumerably Representable (RER), which implies that it consists of computable functions. Thus, some of the results in the below tables are inherited via these inclusions.

Uniform proper Learning

	Any class	RE class	DR class
Realizable	PAC $\not\Rightarrow$ CPAC Theorem 9	PAC $\Rightarrow$ CPAC Theorem 10	PAC $\Rightarrow$ CPAC implied by Theorem 10
Agnostic	PAC $\not\Rightarrow$ CPAC implied Theorem 11	PAC $\not\Rightarrow$ CPAC implied by Theorem 11	PAC $\not\Rightarrow$ CPAC Theorem 11

Uniform improper Learning

	Any class	RE class	DR class
Realizable	open	PAC $\Rightarrow$ CPAC implied by Theorem 10	PAC $\Rightarrow$ CPAC implied by Theorem 10
Agnostic	open	open	open

For the case of non-uniform learning, our Theorem 18 shows that there is a general class (actually the class of all computable functions) that is learnable, but not with a computable learner. We note that this had been shown earlier with a different proof (Soloveichik, 2008). This negative result holds in the realizable case for any (not necessarily proper) learner. Whether non-uniform learnability implies non-uniform learnability with a computable learner for more restricted classes (for example DR or RER classes) is currently open.

4. Proper Learning

We start by considering the proper learning setup and show that, in general, even in the realizable case, PAC learnability does not imply CPAC learnability.

Theorem 9 *There exists a class of computable functions that has VC-dimension 1 and is not* proper *CPAC learnable (even in the realizable setup).*

Proof Let the domain set X be the set of natural numbers, $\mathbb{N}$. Fix a recursive enumeration of all Turing machines $(T_i)_{i\in\mathbb{N}}$. For every $i \in \mathbb{N}$, define h_i as follows:

$$h_i(j) = \begin{cases} 1 & \text{if } j = 2i \\ 1 & \text{if } j = 2i+1 \text{ and the } i\text{'th Turing machine halts on the empty input} \\ 0 & \text{if } j = 2i+1 \text{ and the } i\text{'th Turing machine does not halt on the empty input} \\ 0 & \text{otherwise} \end{cases}$$

All of these functions have finite support, i.e. X_{h_i} is finite for any h_i. Therefore all h_i are computable functions. Finally, let $\mathcal{H}_{halting}$ be the class $\{h_i : i \in \mathbb{N}\}$. Note that VC-dimension$(\mathcal{H}_{halting}) = 1$, since the functions have disjoint support.

Assume by way of contradiction that there exists a computable proper learner for $\mathcal{H}_{halting}$. Consider the probability distribution over $X \times \{0, 1\}$ that assigns all the probability mass to $(2i, 1)$ and 0 to all other points. It is easy to see that for every $i \in \mathbb{N}$, on input training sample $(2i, 1)$ such a learner must output the function h_i. However, since we require CPAC learners to output classifiers using a representation that allows for evaluating them on every domain point, such a CPAC learner would provide a solution to whether Turing Machine T_i halts on the empty input by testing the output of h_i on the test point $2i + 1$. This is a contradiction because this implies we computably solve the halting problem, which is obviously not possible. ■

Note that, while the class of functions $\mathcal{H}_{halting}$ in the proof of the above theorem consists of computable functions (see Remark 7), the class is not RER or DR in the sense of Definitions 5 and 6. We next show, that, at least in the realizable case, if a class is RER, then a class is PAC learnability implies CPAC learnability. Recall that, without computability requirements, a hypothesis class is PAC learnable if and only if it has finite VC-dimension, and in this case, every empirical risk minimizing (ERM) learner is a successful PAC learner (see, for example, Theorem 6.7 in the textbook Shalev-Shwartz and Ben-David (2014)).

Theorem 10 *Every recursively enumerably representable (RER) hypothesis class $\mathcal{H}$ that has finite VC-dimension is* proper *CPAC learnable (in the realizable case).*

Proof We will argue that for such classes there is an algorithm that implements ERM (empirical risk minimization) with respect to this class. Namely, given any realizable labeled sample S, run an algorithm that generates all members of $\mathcal{H}$ one by one, and, for each, compute its empirical error on S. Since we are in the realizable setting, we are guaranteed to eventually find some $h \in \mathcal{H}$ with zero empirical error. As soon as we reach such an h, we halt and h is the output of the learner. ■

Next, we will show that the above result crucially relies on the realizability assumption. We show that even if a hypothesis class meets the stronger requirement of being decidably representable (DR), PAC learnability does not imply proper CPAC learnability in the agnostic case.

Theorem 11 *There exists a decidably representable (DR) hypothesis class of finite VC-dimension (of functions with finite support) that is not* proper *CPAC agnostically learnable.*

Proof Let the domain set X be the set of natural numbers, $\mathbb{N}$. Fix a proof system for first order logic over a rich enough vocabulary that is sound and complete (i.e., every first order formula of

that language has a proof if and only if it is a logical truth). By "rich enough vocabulary" we mean a finite set of functions symbols and relation symbols so that the set of all its logical truths is undecidable, for example a language for the natural numbers with the ordering, addition, and multiplication. For every $i, j \in \mathbb{N}$, we define a function h_{ij} as follows:

$$h_{ij}(k) = \begin{cases} 1 & \text{if } k = 2i \\ 1 & \text{if } k = 2j+1 \\ 0 & \text{otherwise} \end{cases}$$

Enumerate all proofs and logical statements of the proof system. Let

$$\mathcal{H}_{LT} = \{h_{ij} : \text{The } i\text{'th proof in our logic is a proof for the } j\text{'th formula}\}.$$

Note that $\mathcal{H}_{LT}$ is a decidable class. Each function in $\mathcal{H}_{LT}$ has a finite support, and we may therefore assume a list representation for these functions (see Remark 7). Now, to decide if some finite support function is in $\mathcal{H}_{LT}$, we first check if the function maps exactly one even natural number and one odd natural number to 1 and maps all other natural numbers to 0. Then we check if the proof corresponding to the even number, proves the logical statement corresponding to the odd number. Since every function in $\mathcal{H}_{LT}$ assigns the value 1 to only two inputs, $\mathrm{VC}(\mathcal{H}_{LT}) \leq 2$.

Now we will argue that this class is not agnostic CPAC learnable. By way of contradiction, assume that there exists a proper CPAC learner for $\mathcal{H}_{LT}$ in the agnostic setting. For every $i \in \mathbb{N}$, consider a distribution D_i that has all probability mass on the point $(2i+1, 1)$. Thus, every training sample from this distribution is a sequence containing only this point with label 1. Note that if statement $2i+1$ is a logical truth, then a CPAC learner needs to output function $h_{j,2i+1}$, there proof j certifies the truth of statement $2i+1$. Otherwise, the CPAC learner may output any function from the class (since all functions in $\mathcal{H}_{LT}$ have error 1 in that case.

Thus, by feeding this learner a sequence of points $(2i+1, 1)$ and observing the label of the output predictor of this learner on points $2i+1$, we developed a computable procedure to decide whether a statement is a logical truth, in contradiction to our assumption on the richness of the language. ■

4.1. Computable (approximations to) empirical risk minimization

Note that the class $\mathcal{H}_{LT}$ from the proof of Theorem 11 is proper CPAC learnable in the realizable case. Since it is DR, it is also RER, and this thus follows by Theorem 10. In particular, observe that in the realizable case, an input sample point $(2i+1, 1)$ can only occur if the i'th formula is a logical truth. The learner can then go over all finite sequences in lexicographic order until it finds a proof for the i'th formula. Similarly, the argument in the proof of Theorem 10 tells us that if a class is RER, then the recursive enumerability allows us to search for an ERM hypothesis in the class if we are given a promise that there exists a hypothesis in $\mathcal{H}$ with zero empirical error. However, lacking such a promise in the agnostic case, the search is not guaranteed to terminate, leading to the negative conclusions for $\mathcal{H}_{LT}$ in Theorem 11

In this subsection we explore intermediate scenarios. Specifically, we identify other conditions under which such a search will terminate. In some cases, this implies computable empirical risk minimization also in the agnostic case.

Theorem 12 *If $\mathcal{H}$ is an RER class and S is a training sample for which the learner is given a promise that*

$$\min_{h\in\mathcal{H}} L_S(h) \leq \epsilon$$

for some $\epsilon \in [0,1)$, then there is an algorithm that outputs an $h \in \mathcal{H}$ with $L_S(h) \leq \epsilon$.

Proof Similarly to the realizable case, an algorithm can iterate over the hypothesis class, and evaluate the empirical loss of every h on S of every hypothesis and terminate, when it finds a hypothesis h for which $L_S(h) \leq \epsilon$ holds. It is guaranteed to halt by the promise in the statement that such an $h \in \mathcal{H}$ exists. ■

The above result does not guarantee computable empirical risk minimization (as long as there is no promise on the actual minimal empirical risk). However, if $\mathcal{H}$ is RER and has finite VC-dimension, and we are given a bound α on the approximation error of the class (that is, we know $\inf_{h\in\mathcal{H}} L_P(h) \leq \alpha$) and, in addition, we required the learner to halt and output a hypothesis only with high probability over the sample, then we can use the above result to find a hypothesis in $\mathcal{H}$ that has error close to α. In that case, by standard uniform convergence results for classes of finite VC-dimension, we have

$$\min_{h\in\mathcal{H}} L_S(h) \leq \alpha + \epsilon$$

with high probability over the draw of sample S and in this case Theorem 12 yields a hypothesis h with $L_P(h) \leq \inf_{h\in\mathcal{H}} L_P(h) + 2\epsilon$.

Next, we present a situation, in which ERM is computably possible even in the (fully) agnostic case. Note that the condition below holds, for example for so-called *maximum classes* (Floyd and Warmuth, 1995), in which the number of behaviors on each finite domain subset achieves the bound in Sauer's lemma (see Lemma 6.10 in Shalev-Shwartz and Ben-David (2014)). Recall that, for a hypothesis h, we let $X_h \subseteq X$ denote the domain points that h labels with 1.

Theorem 13 *Let $\mathcal{H}$ be an RER class such that for every finite domain subset $W \subseteq X$, the number of subsets of the form $W \cap X_h$ for $h \in \mathcal{H}$ can be computed. Then there is an algorithm that computes ERM for $\mathcal{H}$ (even in the non-realizable case). In particular, if $\mathcal{H}$ in addition has finite VC-dimension, then $\mathcal{H}$ is (agnostically) CPAC learnable.*

Proof Given a sample $S = ((x_1, y_1), \ldots, (x_n, y_n))$, we let $S_X = \{x_1, \ldots, x_n\}$ denote the set of (unlabeled) domain points that occur in S. Since $\mathcal{H}$ is RER, there is an algorithm that enumerates functions in $\mathcal{H}$ and, for each can then compute the intersection $S_X \cap X_h$. Similar to the arguments in the proofs of Theorems 10 and 12, the number of subsets of the form $S_X \cap X_h$ gives a stopping criterion for the search for an ERM. ■

We immediately get the following corollary for maximum classes:

Corollary 14 *If $\mathcal{H}$ is an RER class, has bounded VC-dimension and is a maximum class, then it is CPAC learnable.*

Finally, we show that for classes of initial segments over some partial ordering are CPAC learnable even in the agnostic case.

Theorem 15 *If $\mathcal{H}$ is the class of all initial segments under any (partial) ordering relation (containing also the constant 0 hypothesis), and $\mathcal{H}$ is RER then there is a CPAC ERM learner for $\mathcal{H}$ (even in the agnostic case).*

Proof First, note that, since $\mathcal{H}$ is a class of initial segments, the VC-dimension of $\mathcal{H}$ is 1.

Let $\prec$ denote the partial order, and for $x \in X$, let $\text{init}(x)$ denote the initial segment corresponding to x, that is $\text{init}(x) = \{y \in X \mid y \prec x\}$. Since $\mathcal{H}$ consists of initial segments, for each $h \in \mathcal{H}$, there is an $x_h \in X$ such that $X_h = \text{init}(x_h)$ (recall that X_h denotes the set of all domain points that are labeled 1 by h). And since we assume that $\mathcal{H}$ is the class of *all* initial segments over $\prec$, we get

$$\mathcal{H} = \{\text{init}(x) \mid x \in X\}.$$

Now note that for every pair of points $(x, y) \in X^2$, the class $\mathcal{H}$ has exactly three behaviors. If x and y are comparable by $\prec$, say without loss of generality $x \prec y$, then $\mathcal{H}$ contains the labelings $(0,0), (0,1)$ and $(1,1)$ for (x, y). If x and y are not comparable, then $\mathcal{H}$ contains the labelings $(0,0), (0,1)$ and $(1,0)$ for (x, y).

This, again, can be turned into a stopping criterion for a search for an empirical risk minimizer. Given a sample S, there is an algorithm enumerates all functions in $\mathcal{H}$, until for each pair of domain points in S, three labeling behaviors were observed. Then, it can compute the empirical error of each of these functions over S and output and ERM hypothesis $h \in \mathcal{H}$. Since the VC-dimension of $\mathcal{H}$ is 1, this is a CPAC learner for $\mathcal{H}$. ■

5. Non-uniform Learning

We now consider the non-uniform learning setup (recall Definition 3), and show that, here as well, there are classes that are learnable (when learners can be arbitrary functions), not by a computable learner. A similar result had been shown in Soloveichik (2008), however, we provide a different proof. We first note that a hypothesis class $\mathcal{H}$ is non-uniformly learnable if and only if it is a countable union of finite VC-classes.

Theorem 16 (Theorem 7.2 in Shalev-Shwartz and Ben-David (2014)) *$\mathcal{H}$ is non-uniformly learnable if and only if there exist a sequence of classes $(\mathcal{H}_i)_{i\in\mathbb{N}}$, each of finite VC-dimension, such that $\mathcal{H} = \bigcup_{i\in\mathbb{N}} \mathcal{H}_i$.*

Corollary 17 *The class $\mathcal{H}_{comp}$ of all computable functions from $\mathbb{N}$ to $\{0,1\}$ is non-uniformly learnable.*

Proof Since each function h in $\mathcal{H}_{comp}$ is computable, there exists a program computing h. Thus, $\mathcal{H}_{comp}$ is countable, and therefore Theorem 16 applies. ■

However, we will show that the class $\mathcal{H}_{comp}$ cannot be learned by any computable learner (proper or non-proper).

Theorem 18 (Also see Soloveichik (2008)) *No computable learner can non-uniformly learn the class $\mathcal{H}_{comp}$.*

We will now provide an alternative proof for this result. We start by proving a "computable version" of the classic No-Free-Lunch Theorem, which states that for every learner, for a fixed sample size m, there exists a (simple) distribution over $2m$ domain points, on which this learner will suffer a high expected loss. We show that, if the learner is computable, there is also an algorithm that finds this distribution. The Lemma and proof below are modifications of Theorem 5.1 in Shalev-Shwartz and Ben-David (2014).

Lemma 19 (Computable No-Free-Lunch) *For any computable learner A, and for any $m \in \mathbb{N}$, any domain X of size at least $2m$, any subset $\{x_1, \ldots, x_{2m}\} \subseteq X$ of size $2m$, we can computably find a function $f : \{x_1, \ldots, x_{2m}\} \to \{0, 1\}$ such that*

$$\mathrm{E}_{S\sim D^m}[L_D(A(S))] \geq \frac{1}{4}, \qquad \textit{and} \qquad \mathrm{Prob}_{S\sim D^m}\left[L_D(A(S)) \geq \frac{1}{8}\right] \geq \frac{1}{7},$$

where D is a uniform distribution over $\{(x_1, f(x_1)), \ldots, (x_{2m}, f(x_{2m}))\}$.

Proof The existence of f and D with the stated properties follow from the original No-Free-Lunch Theorem. All that is left to show here, is that we can computably find the function f.

There are $T = 2^{2m}$ functions from $\{x_1, \ldots, x_{2m}\}$ to $\{0, 1\}$, which we denote by $f_1, \ldots, f_T$. For each $1 \leq i \leq T$, define D_i to be the uniform distribution over $\{(x_1, f_i(x_1), \ldots, (x_{2m}, f_i(x_{2m}))\}$ We know one of these are the function f and distribution D that we are looking for.

For each $1 \leq i \leq T$, there are $k = (2m)^m$ possible sequences of m samples drawn from the distribution D_i. Let us denote these sequences as $S_1^i, \ldots S_k^i$. Each of these sample set occurs with equal probability. Since A is computable, we can calculate the expected loss of the learner with respect to distribution D_i as follows:

$$\mathrm{E}_{S\sim D_i^m}[L_{D_i}(A(S))] \;=\; \frac{1}{k}\sum_{j=1}^{k} L_{D_i}(A(S_j^i)) \;=\; \frac{1}{k}\sum_{j=1}^{k}\left[\frac{1}{2m}\sum_{l=1}^{2m}\mathbf{1}[A(S)(x_l) \neq f_i(x_l)]\right]$$

From the No-Free-Lunch theorem, we know that one of the expected losses is at least $\frac{1}{4}$. We can compute this quantity for each i from 1 to T until we find the quantity at least $\frac{1}{4}$. This gives the function f that we seek. The high probability statement follows from the statement about high expected loss (see Shalev-Shwartz and Ben-David (2014) for details). ■

With this, we proceed to prove that $\mathcal{H}_{comp}$ is not learnable by a computable learner.

Proof [Proof of Theorem 18] Assume by way of contradiction that there is a computable learner A that non-uniformly learns $\mathcal{H}_{comp}$. By Definition 3, this means that for any computable function h, and any $\epsilon, \delta \in (0, 1)$, there is a natural number $n_A(\epsilon, \delta, h)$ such that for any distribution D and any $n \geq n_A(\epsilon, \delta, h)$, we get

$$\mathrm{Prob}_{S\sim D^n}\left[L_D(A(S)) \leq L_D(h) + \epsilon\right] \geq 1 - \delta.$$

Consider the following partition of the natural numbers $\mathbb{N} = \cup_{i\in\mathbb{N}} P_i$ where each $P_i = [l_i, u_i]$. For each i, $u_i - l_i + 1 = 10i$, $l_{i+1} = u_i + 1$ and $l_1 = 1$. For each $i \in \mathbb{N}$ and for $m_i = 5i$, by Lemma 19, there exists a function f_i and a distribution D_i on P_i such that for the distribution D_i' over $\mathbb{N} \times \{0, 1\}$, the learner A fails to uniformly learn for $\epsilon = \frac{1}{8}$ and $\delta = \frac{1}{8}$. Here, D_i' is such that

$$D_i'((x, y)) = \begin{cases} D_i((x, y)) & \text{if } x \in P_i \\ 0 & \text{otherwise} \end{cases}$$

Consider a function $h_0 : \mathbb{N} \to \{0,1\}$ such that, for any $x \in \mathbb{N}$, $h_0(x) = f_j(x)$ where j is such that $x \in P_j$ (we can compute such a j given an x). Since each f_i can be found computably, h_0 is a computable function. Hence $h_0 \in \mathcal{H}_{comp}$.

There is a $k \in \mathbb{N}$ such that $|P_k| > 2n_A(\frac{1}{8}, \frac{1}{7}, h_0)$. For the distribution D'_k, with probability at least $\frac{1}{7}$ over samples of size $2n_A(\frac{1}{8}, \frac{1}{7}, h_0)$ drawn from D'_k, $L_{D'_k}(A(S)) > \frac{1}{8}$. This contradicts our assumption that A non-uniformly learned the class $\mathcal{H}_{comp}$. ■

Remark 20 *Similar impossibility results as those in Theorem 18, can be shown for certain subclasses of the class $\mathcal{H}_{comp}$. For example, the same proof technique will show that the class of all computable functions with a bounded run time cannot non-uniformly be learned by a learner with an accordingly bounded run time.*

6. Improper Learning

To understand the implications of CPAC-learnability in the more general setup of non-proper learners, we obtain the following reduction inspired by a reduction in Daniely et al. (2014).

For a hypothesis class $\mathcal{H}$ and a sequence $S = ((x_1, y_1), (x_2, y_2), \ldots (x_m, y_m))$ of labeled domain points, we say that S has a *realizable labeling* if there is an $h \in \mathcal{H}$ such that $y_i = h(x_i)$ of all pairs $(x_i, y_i) \in S$. We say that S has a *random labeling* if each y_i was the result of a random coin flip where $\text{Prob}(y_i = 1) = \text{Prob}(y_i = 0) = \frac{1}{2}$. With this terminology, we introduce the following task.

Definition 21 (The Distinguishing Problem) *Given a hypothesis class $\mathcal{H}$, we say that a (potentially randomized) function $A : \bigcup_{n\in\mathbb{N}}(X \times \{0,1\})^n \to \{$"realizable", "unrealizable"$\}$ solves* the distinguishing problem *if for any $\delta > 0$, there is an $M \in \mathbb{N}$ such that for any sequence $S = (x_1, \ldots, x_M) \in X^M$, we have*

- *for T - a realizable labeling of S, $Pr(A(T) =$ "realizable"$) \geq 1 - \delta$,*
- *for T - a random labeling of S, $Pr(A(T) =$ "unrealizable"$) \geq 1 - \delta$,*

where the probability is taken over the randomization of A and the random coin flips that generated the labels in the latter case.

Theorem 22 *If there is an (agnostic) CPAC learner of a class $\mathcal{H}$ whose range is a class of finite VC-dimension, we can solve the distinguishing problem for $\mathcal{H}$ with a computable distinguisher.*

Proof We let F be a CPAC learner that learns the class $\mathcal{H}$ and has a range with finite VC-dimension. We let $\mathcal{H}'$ denote this class of output hypotheses of F, and let d denote the VC-dimension of $\mathcal{H}'$.

By the Sauer's Lemma, we know that the number of behaviors $\mathcal{H}'$ exhibits on a set of size m is bounded by a function $p(m)$ that is polynomial in m. We will now show that we can construct an algorithm A to solve the distinguishing problem for $\mathcal{H}$.

For a given $\delta > 0$, choose $M > d$ and large enough so that $p(M)\left(\frac{e}{4}\right)^{\frac{M}{4}} < \delta$. Here $p(M) = \left(\frac{eM}{d}\right)^d$ which is an upper bound to the growth function of $\mathcal{H}'$ for M.

For a labeled sequence $S = ((x_1, y_1), \ldots, (x_M, y_M)) \in (X \times \{0,1\})^M$, we let D_S denote the uniform distribution over S. Now we describe an algorithm A that solves the distinguishing problem. Given S, the algorithm A will

1. By uniformly sub-sampling from S, generate a sample from distribution D_S to run learner F with error parameter $\frac{1}{4}$ and confidence parameter δ.
2. Compute $L_S(h)$ output hypothesis h of F.
3. If $L_S(h) < \frac{1}{4}$, return "realizable". Else return "unrealizable".

Note that for a realizable labeling S, we are guaranteed $L_S(h) < \frac{1}{4}$ with probability at least $1 - \delta$.

The set of hypotheses that F may return has at most $p(M)$ behaviors on any set of instances of size M. The probability that any behavior yields error less than $\frac{1}{4}$ on S is bounded by

$$p(M) \cdot \frac{\binom{M}{\frac{M}{4}}}{2^M} \leq p(M) \cdot \left(\frac{e}{4}\right)^{\frac{M}{4}} < \delta$$

So, with probability at least $1 - 2\delta$ (over the random coin flips and the sampling from D_S), the output hypothesis makes more than $\frac{1}{4}$ error on S and is hence identified as "unrealizable" with at least this probability. ■

Observe that the classes $\mathcal{H}_{halting}$ and $\mathcal{H}_{LT}$ that we used for the negative results on proper CPAC learning in Section 4, are actually CPAC learnable if we remove the properness requirement. All functions in these classes map at most two domain-points to 1 and the rest of the domain to 0. They are thus both subclasses of $\mathcal{H}_2$, the class of all hypotheses h with $|X_h| \leq 2$. This class is actually properly CPAC learnable, thus both $\mathcal{H}_{halting}$ and $\mathcal{H}_{LT}$ are (improperly) CPAC learnable. This leads us our first conjecture:

Conjecture 23 *If a class $\mathcal{H}$ is (improperly) CPAC learnable, then there is a superclass $\mathcal{H}' \supseteq \mathcal{H}$ such that $\mathcal{H}'$ is proper CPAC learnable.*

Theorem 22, along with Conjecture 23 leads to the following corollary.

Corollary 24 *If $\mathcal{H}$ is (improper) CPAC learnable, then $\mathcal{H}$ is computably distinguishable.*

We also conjecture the following:

Conjecture 25 *There is a class $\mathcal{H}$ consisting of computable hypotheses, and with finite VC-dimension, such that $\mathcal{H}$ is not computably distinguishable.*

Corollary 24, along with Conjecture 25 would imply the existence of a class $\mathcal{H}$ of finite VC-Dimension that is not (improperly) CPAC learnable.

7. Conclusion and Future Work

In this paper we have initiated an investigation of analyzing the statistical notions of learnability under the natural requirement that learner should be a computable function. We have shown that the addition of such an assumption disrupts the fundamental characterization of learnability by the finite VC-dimension of a class (even when the class is RE and every function in it is computable).

However, many question remain open. In particular we were not able to resolve whether decidable classes $\mathcal{H}$ with finite VC-dimension are necessarily improperly CPAC learnable in the agnostic case. A characterization of learnability by computable learners which are not necessarily proper seems to be the most interesting follow-up open question.

Furthermore we don't have a clear characterization of when a class is properly CPAC learnable in the agnostic case. So far, we have seen that proper CPAC learning always works in the realizable case for recursively enumerable classes $\mathcal{H}$ with finite VC-dimension. However it would be interesting to find out in which cases this would be true for the agnostic case.

References

Shai Ben-David, Pavel Hrubes, Shay Moran, Amir Shpilka, and Amir Yehudayoff. A learning problem that is independent of the set theory ZFC axioms. *CoRR*, abs/1711.05195, 2017. URL http://arxiv.org/abs/1711.05195.

Shai Ben-David, Pavel Hrubes, Shay Moran, Amir Shpilka, and Amir Yehudayoff. Learnability can be undecidable. *Nature Machine Intelligence*, 1:44–48, 2019.

Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Symposium on Theory of Computing, STOC*, pages 441–448, 2014.

Sally Floyd and Manfred K. Warmuth. Sample compression, learnability, and the vapnik-chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.

David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

David Soloveichik. Statistical learning of arbitrary computable classifiers. *arXiv preprint arXiv:0806.3537*, 2008.

Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 2000.