On the Learnability of the Uncomputable

Richard H. Lathrop
Information and Computer Science Dept.
University of California
Irvine, CA 92717–3425
rickl@ics.uci.edu

In *Proceedings of the 13th International Conference on Machine Learning,* pp. 302–309,
Bari, Italy, 3–6 July 1996, ed. Saitta, L., Morgan Kaufman Publishers, San Francisco.

Abstract

Within Valiant's model of learning as formalized by Kearns, we show that computable total predicates for two formally uncomputable problems (the classical Halting Problem, and the Halting Problem relative to a specified oracle) are formally learnable from examples, to arbitrarily high accuracy with arbitrarily high confidence, under any probability distribution. The Halting Problem relative to the oracle is learnable in time polynomial in the measures of accuracy, confidence, and the length of the learned predicate. The classical Halting Problem is learnable in *expected* time polynomial in the measures of accuracy, confidence, and the $(1-\epsilon/16)^{th}$ percentile length and run-time of programs which *do* halt on their inputs (these quantities are always finite). Equivalently, the mean length and run-time may be substituted for the percentile values in the time complexity statement. The proofs are constructive. *While the problems are learnable, they are not polynomially learnable, even though we do derive polynomial time bounds on the learning algorithm.*

1 INTRODUCTION

Can we learn what we cannot compute? We will show that in at least some cases the answer is "yes." The widely accepted formal model for computation is the Turing machine (TM), and the Halting Problem is the canonical uncomputable task (Turing 1936). Valiant (Valiant 1984) proposed a formal analysis of learning as the phenomenon of knowledge acquisition in the absence of explicit programming. The methodology was systematically organized into a consistent and rigorous framework by Kearns (Kearns 1990), which we follow. We consider two uncomputable problems and show that they are learnable within this formal model of learnability. *While the problems are learnable, they are not polynomially learnable, even though we do derive polynomial time bounds on the learning algorithm.*

1.1 ORGANIZATION

Section 2 introduces some mathematical preliminaries. Section 3 states a problem that incorporates an oracle and is uncomputable because the Halting Problem reduces to it. We present a learning algorithm, and prove that it formally learns a predicate for this problem. The algorithm runs in time polynomial in the measures of accuracy $(1/\epsilon)$, confidence $(1/\delta)$, and the size of the learned predicate. Section 4 presents a learning algorithm for the classical Halting Problem. We prove that the learning algorithm always terminates, and with arbitrarily high confidence produces an arbitrarily accurate total predicate, under any probability distribution. The algorithm runs in *expected* time polynomial in the measures of accuracy, confidence, and the $(1-\epsilon/16)^{th}$ percentile length and run-time of programs which *do* halt on their inputs (these quantities are always finite). If desired, the mean length and run-time may be substituted for the percentile values in the time complexity statement. Finally, section 5 states some important limitations on our results. Major limitations are an impoverished knowledge representation capability, and the fact that positive instances are *a priori* known to be classified relative to the Halting Problem (these correspond to severe restrictions on what Kearns (Kearns 1990) terms the hypothesis class and the target class, respectively).

1.2 BACKGROUND

Let C (the target class) and H (the hypothesis class) be representation classes over X (the *domain,* or *instance space*). Then C is *learnable from examples by* H (Kearns 1990, p. 11) if there is an algorithm $\mathcal{A}$ with access to POS and NEG, taking inputs ϵ (the accuracy parameter, $0 < \epsilon < 1$) and δ (the confidence parameter, $0 < \delta < 1$), such that for any $c \in C$, any probability distributions $\mathbf{D}^+$ and $\mathbf{D}^-$, and any inputs ϵ and δ, algorithm $\mathcal{A}$ halts and outputs a representation $h_A \in H$ that with probability greater than $(1-\delta)$ has an accuracy greater than $(1-\epsilon)$. C is *learnable from examples* if there exists a representation class H such that C is learnable from examples by H. In this paper, then, when we say that an uncomputatable problem (such as the Halting Problem) is learnable from examples, we mean that there exists a representation class C that is learnable from examples, and some $c \in C$ represents the uncomputatable problem's characteristic set. Thus the learned total predicate probably approximates the uncomputable predicate.

If, further, C and H are polynomially evaluatable, and if $\mathcal{A}$ runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|c|$, then C is *polynomially learnable from examples by* H. By separating the definition of learnability from statements about time complexity, Kearns is able to formally discuss and compare algorithms across a spectrum of complexities (for example, his analysis encompasses the super-polynomial but sub-exponential learning algorithm by Ehrenfeucht and Haussler (Ehrenfeucht and Haussler 1988) for decision trees with at most a fixed polynomial number of nodes). *Note that by definition no uncomputable problem can possibly be polynomially learnable, because an uncomputable target representation cannot possibly be polynomially evaluatable.*

1.3 INTUITION

How can something that is not even computable be learnable? Isn't there a contradiction here?

The key insight exploits the "probably approximately" aspect of the formal learning model. The learned hypothesis need not compute the Halting Problem (which of ocurse would be impossible). Rather, it need compute only a total predicate that, with high likelihood, agrees with the Halting Problem on a high percentage of instances.

For any distribution and any desired accuracy $(1-\epsilon)$, there are infinitely many total predicates that agree with the Halting Problem on at least $(1-\epsilon)$ of the probability-weighted instances (program-instance pairs). In order to succeed, the learner need find only one of them, and the proofs below show that this is possible. Thus, the learner learns a computable hypothesis that only *approximates* the Halting Problem.

The basic proof strategy exploits the fact that the probability distribution $\mathbf{D}^+$ on positive instances induces a probability distribution on their run-times. For any distribution there is some (appropriately large) threshold such that at least $(1-\epsilon)$ of the probability-weighted instances terminate before the threshold. The proofs below show how to estimate the threshold by sampling POS. Of course, the learner may be unlucky enough to draw short-running instances during training or long-running instances during testing. For any desired confidence $(1-\delta)$, the threshold can be chosen high enough that the probability of such an unlucky case is at most δ. Thus, the learned computable hypothesis *probably* approximates the Halting Problem.

1.4 RELATED WORK

The literature in both computability and learnability spans decades (Blum and Blum 1975; Gold 1967; Herken 1988; Turing 1936). The reader is referred to references (Cutland 1980; Hopcroft and Ullman 1979; Lewis and Papadimitriou 1981; Minsky 1967) for theory of computation and to references (Angluin and Smith 1983; Kearns 1990; Valiant 1984; Pitt 1990) for formal machine learning, among others. The formal approach has been widely explored by the machine learning community (Pitt 1990), and a great deal is known about necessary conditions, limitations, and bounds (Blumer et al. 1989; Blumer et al. 1987; Ehrenfeucht et al. 1989; Kearns and Valiant 1994; Linial et al. 1991; Pitt and Valiant 1988; Shvaytser 1990). Amsterdam (Amsterdam 1988) discusses limitations of the framework. The problem of language identification in the limit differs from this work in finding an exact rather than probabilistic predicate, and in lacking polynomial time complexity bounds. Gold (Gold 1967) showed that even the class of all regular languages is not inferable from positive data. Gasarch and Smith (Garasch and Smith 1992) consider identification in the limit under a model in which the learner is permitted to ask undecidable questions. Cherniavsky and Smith (Cherniavsky and Smith 1987) investigate the relationship between recursive enumerability, inductive inference and program testing (which includes infinite loops), but do not extend it to

the probabilistic learning considered here. Fulk and Jain (Fulk and Jain 1994) consider approximate inference under which the class of recursive functions is identifiable in the limit, but do not establish polynomial time bounds. Benedek and Itai (Benedek and Itai 1991) consider learnability when the distribution is fixed and known. They mention the possibility that the target class may be uncomputable but do not explicitly consider the case. The Halting Probability Problem (Machlin and Stout 1990), which uses the probability that a randomly drawn TM will halt to investigate randomness in algorithmic information theory, is only distantly related to the present work.

2 MATHEMATICAL PRELIMINARIES

To be of interest, a learning system must perform better than a fair coin-flip. We assume throughout that ϵ and δ are both less than $1/2$.

Notationally, let the data for learning be denoted by pairs $\langle \mathcal{M}, \mathcal{I} \rangle$, where $\mathcal{M}$ is a TM program and $\mathcal{I}$ is its input ("instance" and "example" are synonyms for such a pair). An instance is positive if $\mathcal{M}$ halts on $\mathcal{I}$, otherwise negative. Let $\#(\langle \mathcal{M}, \mathcal{I} \rangle)$ be the number of steps that $\mathcal{M}$ runs on $\mathcal{I}$ (defined for positive examples, and undefined for negative examples), and let $|\langle \mathcal{M}, \mathcal{I} \rangle|$ be the length of the instance as a string of symbols (defined for all examples).

Let r be the inverse cumulative distribution of positive instance run-times induced on $\#(.)$ by the probability density function $\mathbf{D}^+$, defined for $0 < x < 1$ by

$$r(x) = \min_k \left\{ \left(\sum_{\#(\langle \mathcal{M}, \mathcal{I} \rangle) \leq k} \mathbf{D}^+ (\langle \mathcal{M}, \mathcal{I} \rangle) \right) \geq x \right\}$$

Thus $r(x)$ is the minimum number k such that the probability is at least x that a positive instance halts in k steps or less. For example, $r(0.5)$ is the median run-time of positive instances, while 90% halt in $r(0.9)$ steps or less. r is completely determined by $\mathbf{D}^+$, but is unknown to the learner because $\mathbf{D}^+$ is unknown. Let l be defined similarly as the inverse cumulative distribution of positive instance lengths induced on $|.|$ by $\mathbf{D}^+$. Let ρ be the mean of $\#(.)$ on positive examples (their mean run-time) and let λ be the mean of $|.|$ on positive examples (their mean length). ρ and λ are (possibly infinite) distribution-dependent parameters, unknown to the learner.

2.1 IDENTITIES AND INEQUALITIES

Fact 1 $\left(1 + x^{-1}\right)^x < e$, for $0 < x$.

Fact 2 $\left(1 - x^{-1}\right)^x < e^{-1}$, for $0 < x$.

Fact 3 $\exp(-x^{-1}) < x$, for $0 < x$.

Fact 4 $\sum_{i=0}^{\infty} \left(\prod_{j=1}^{m} (i+j) \right) x^i = \frac{m!}{(1-x)^{m+1}}$
for m a non-negative integer and $0 < x < 1$.

Fact 5 $\sum_{i=0}^{\infty} x^i i^m < 2^{m+1} m!$
for m a non-negative integer and $0 < x < 1/2$.

Fact 6 $\sum_{i=1}^{n} i^m = \sum_{j=0}^{m+1} c_j n^j$
where m and n are non-negative integers and the coefficients c_j are given by

$$\begin{aligned} c_0 &= 0 \\ c_j &= \left[\binom{m}{j-1} - \sum_{k=j+1}^{m+1} c_k \binom{k}{j-1} \right] / j \end{aligned}$$

Fact 7 (Chernoff Bound (Chernoff 1952), (Valiant 1984)) In n independent trials, each with probability of success at least p, the probability that there are at most k successes, where $k < np$, is at most $\left(\frac{n-np}{n-k}\right)^{n-k} \left(\frac{np}{k}\right)^k$

2.2 LEMMAS

Lemma 1 In Fact 7, if $p = (1 - b\epsilon)$ and $k = n(1 - c\epsilon)$ where $0 < b < c \leq 1$, then the probability that there are at most k successes is at most[1] $[\exp(c - \ln c - (b - \ln b))]^{n\epsilon}$.

Lemma 2 In Fact 7 above, if $p = b\epsilon$ and $k = nc\epsilon$ where $0 < c < b \leq 1$, then the probability that there are at most k successes is also at most $[\exp(c - \ln c - (b - \ln b))]^{n\epsilon}$.

Lemma 3 Let $f(\langle \mathcal{M}, \mathcal{I} \rangle)$ be any function from instances to non-negative numbers, let g be the inverse cumulative distribution induced on $f(.)$ by $\mathbf{D}^+$, and let γ be the mean of f. Then for $0 < \epsilon \leq 1$, $g(1 - \epsilon) \leq \gamma \epsilon^{-1}$.

Corollary $r(1 - \epsilon) \leq \rho \epsilon^{-1}$.

Corollary $l(1 - \epsilon) \leq \lambda \epsilon^{-1}$.

Lemma 4 Let f, g, and γ be as in Lemma 3. Then $g(1 - \epsilon)$ is finite for $0 < \epsilon \leq 1$ (even if γ is infinite).

Corollary $r(1 - \epsilon)$ is finite for $0 < \epsilon \leq 1$.

[1] Throughout this paper ln denotes the natural logarithm and $\log_2$ the logarithm to the base 2.

Corollary $l(1-\epsilon)$ is finite for $0 < \epsilon \leq 1$.

3 THE HALTING PROBLEM RELATIVE TO AN ORACLE

In this section we precisely state a computational problem and prove that it is uncomputable. We then state an algorithm and prove that it learns a total predicate for the problem, achieving arbitrarily high accuracy with arbitrarily high confidence under any probability distribution. The time complexity of the learner is polynomial in the measures of accuracy, confidence, and the length of the learned predicate.

The problem consists of the Halting Problem relative to an oracle. For input instance $\langle \mathcal{M}, \mathcal{I} \rangle$, the task is to output a 1 if $\mathcal{M}$ halts when given $\mathcal{I}$ as input and a 0 otherwise, and then halt. The learner (but not the solution) may make use of a routine, ORACLE, that accepts two arguments: an instance $\langle \mathcal{M}, \mathcal{I} \rangle$, and an integer S governing output format. If $\mathcal{M}$ fails to halt when given $\mathcal{I}$ as input, then the call ORACLE($\langle \mathcal{M}, \mathcal{I} \rangle, S$) never returns and so the caller fails to halt. However, if $\mathcal{M}$ halts on $\mathcal{I}$ then the call ORACLE($\langle \mathcal{M}, \mathcal{I} \rangle, S$) returns in unit time, and the returned value is the number of steps $\mathcal{M}$ takes before halting on $\mathcal{I}$. For technical reasons in the proof below, we assume that the input format for ORACLE is the same as is returned by calls to POS, and that the format of the returned value is a binary integer, written least significant bit first onto every S^{th} square of the TM tape, starting at the head position at the time of the call.

It is easy to see that the Halting Problem relative to ORACLE is uncomputable, because if some TM, say $\mathcal{M}_{\mathcal{O}}$, could compute it, then we could easily modify $\mathcal{M}_{\mathcal{O}}$ to solve the classical Halting Problem. Let NON-ORACLE be a TM with identical input/output behavior to ORACLE (but a longer run-time), and $\mathcal{M}_{\mathcal{NO}}$ be identical to $\mathcal{M}_{\mathcal{O}}$ except that wherever $\mathcal{M}_{\mathcal{O}}$ calls ORACLE, $\mathcal{M}_{\mathcal{NO}}$ calls NON-ORACLE. Consequently $\mathcal{M}_{\mathcal{NO}}$ will have identical input/output behavior to $\mathcal{M}_{\mathcal{O}}$ (but a longer run-time). Then $\mathcal{M}_{\mathcal{O}}$ computes the Halting Problem relative to ORACLE if and only if $\mathcal{M}_{\mathcal{NO}}$ computes the Halting Problem without an oracle, which we know to be impossible.

3.1 THE LEARNING ALGORITHM

Next we show that the Halting Problem relative to ORACLE is learnable, by exhibiting an algorithm that learns it.

Let $\mathcal{A}_{\mathcal{O}}$ be as follows:

1. Input ϵ and δ.
2. $N \leftarrow 16\epsilon^{-1}\delta^{-1}$.
3. $C \leftarrow \lceil N(1-\epsilon/4) \rceil$.
4. Make N calls to ORACLE(POS(), N).
5. Compute K as the C^{th} smallest returned value.
6. Output a program $\mathcal{L}_{\mathcal{O}}$ that does:
   A. Input an instance $\langle \mathcal{M}, \mathcal{I} \rangle$.
   B. Simulate $\mathcal{M}$ on $\mathcal{I}$ for K steps.
   C. If $\mathcal{M}$ has halted then print 1, else print 0.
   D. Halt.
7. Halt.

3.2 PROOF THAT $\mathcal{A}_{\mathcal{O}}$ LEARNS THE HALTING PROBLEM RELATIVE TO ORACLE.

Theorem 1 Let $\mathcal{A}_{\mathcal{O}}$ be as stated. Then:

1.A $\mathcal{L}_{\mathcal{O}}$ has an error rate of 0 on negative examples;

1.B with probability greater than $(1-\delta)$, $\mathcal{L}_{\mathcal{O}}$ has an accuracy greater than $(1-\epsilon)$ on positive examples; and

1.C $\mathcal{A}_{\mathcal{O}}$ halts in time polynomial in ϵ^{-1}, δ^{-1}, and the size of the learned predicate.

Proof:

1.A $\mathcal{L}_{\mathcal{O}}$ has an error rate of 0 on negative examples because it classifies as positive only those instances which do halt (in K or fewer steps). □ (1.A)

1.B By definition of r, the learned predicate $\mathcal{L}_{\mathcal{O}}$ has an accuracy on future positive examples greater than $(1-\epsilon)$ if and only if $\mathcal{A}_{\mathcal{O}}$ terminates with $K \geq r(x)$ for some $x > (1-\epsilon)$. We show that with probability greater than $(1-\delta)$, $\mathcal{A}_{\mathcal{O}}$ terminates with $K \geq r(1-\epsilon/2)$.

Term a call to ORACLE(POS(), N) a success when its returned value equals or exceeds the (unknown) value of $r(1-\epsilon/2)$. Then $K \geq r(1-\epsilon/2)$ if the C^{th} smallest run-time was a success, which in turn is true if we have more than $N\epsilon/4$ successes in our N calls. By definition of r the probability of a success is at least $\epsilon/2$. Consequently, by Lemma 2 with $b = 1/2$, $c = 1/4$, and $n = N = 16\epsilon^{-1}\delta^{-1}$, the probability that this condition fails and there are at most $N\epsilon/4$

successes is less than $\exp\left(-1.227 \times \delta^{-1}\right)$, and by Fact 3 this is less than δ. □ (1.B)

1.C Let L be the length of the program that is to be output in step 6. Steps 1-4 and 6-7 are obviously polynomial-time in ϵ^{-1}, δ^{-1}, and L. However, in step 5 we must compute the C^{th} smallest of the values returned by the N calls to ORACLE(POS(), N), and these values may be arbitrarily huge.

By making the N calls to ORACLE(POS(), N) with the tape head on consecutive adjacent squares of the tape, we may arrange that the bits of the N returned values are interdigitated in increasing order of significant bits:

$$\ldots \not\flat\, d_{11}d_{12}d_{13}\ldots d_{1N}d_{21}d_{22}\ldots d_{2N}d_{31}d_{32}\ldots \not\flat\, \not\flat \ldots$$

where $\not\flat$ is the blank symbol initializing the tape and d_{ij} is the i^{th} least significant bit of the j^{th} call, or $\not\flat$ if the j^{th} call returned fewer than i significant bits.

Now if K is the C^{th} smallest value, we can find K in time polynomial in N and $\log_2 K$. First, start at d_{11} and skip to the right in groups of N bits, at the i^{th} stage examining the N tape squares $d_{ij}, 1 \leq j \leq N$, until at least C of the N squares are $\not\flat$. This occurs at $i_0 = \lfloor \log_2 K \rfloor + 2$. Eliminate all values for which $d_{i_0,j} \neq \not\flat$, and find the C^{th} smallest value of the N or fewer remaining integers, each of size $\lfloor \log_2 K \rfloor + 1$ bits or less. This can easily be done in time polynomial in N and $\lfloor \log_2 K \rfloor$.

Because $\lfloor \log_2 K \rfloor \leq L$ and $N = 16\epsilon^{-1}\delta^{-1}$, step 5 and consequently $\mathcal{A}_{\mathcal{O}}$ runs in time polynomial in ϵ^{-1}, δ^{-1}, and L. □ (1.C)

Consequently $\mathcal{A}_{\mathcal{O}}$ formally learns the Halting Problem relative to ORACLE in time polynomial in ϵ^{-1}, δ^{-1}, and the size of the learned predicate. □ (1)

3.3 A BOOLEAN ORACLE

Traditionally, an oracle returns only a YES/NO answer. The oracle we employed above returned a numeric answer. We can modify the algorithm to accommodate learning with a YES/NO oracle as follows. Let ORACLE-P be defined the same as ORACLE, except that ORACLE-P($\langle \mathcal{M}, \mathcal{I} \rangle, S$) returns YES if $\#(\langle \mathcal{M}, \mathcal{I} \rangle) \leq S$ and NO if $\mathcal{M}$ halts on $\mathcal{I}$ and $\#(\langle \mathcal{M}, \mathcal{I} \rangle) > S$. As with ORACLE, if $\mathcal{M}$ fails to halt on $\mathcal{I}$ then ORACLE-P($\langle \mathcal{M}, \mathcal{I} \rangle, S$) fails to return. Replace statements 4 and 5 of $\mathcal{A}_{\mathcal{O}}$ with the following:

4. For $I = 1$ to N do EXAMPLE[I] $\leftarrow$ POS().

5.a. $K \leftarrow 1$.

5.b. $K \leftarrow 2K$.

5.c. $J \leftarrow 0$.

5.d. $COUNT \leftarrow 0$.

5.e. $J \leftarrow J + 1$.

5.f. If ORACLE-P(EXAMPLE[J], K) then $COUNT \leftarrow COUNT + 1$.

5.g. If $J < N$ then go to 5.e.

5.h. If $COUNT < C$ then go to 5.b.

We leave to the reader the proof that the Boolean version of $\mathcal{A}_{\mathcal{O}}$ learns the Halting Problem relative to ORACLE-P in time polynomial in ϵ^{-1}, δ^{-1}, and the size of the learned predicate. Note that the output format of POS must be adjusted so that the results of successive calls do not over-write each other (for example, by interdigitation as in 1.C), and the input format of ORACLE-P must match.

4 LEARNING TO SOLVE THE HALTING PROBLEM

In this section we turn our attention to learning to solve the classical Halting Problem. The problem statement is identical to that in the previous section, except that the learner is not permitted to use ORACLE. We state an algorithm and prove that it learns a total predicate for the Halting Problem, achieving arbitrarily high accuracy with arbitrarily high confidence under any probability distribution. The *expected* time complexity of the learner is polynomial in the measures of accuracy, confidence, and the mean length and run-time of programs which *do* halt on their inputs.

We state our results in terms of the mean positive instance length and run-time, λ and ρ, because it leads to a more intuitive and compact form. Alternatively, we could have phrased all claims and proofs below in terms of $l(1 - \epsilon/16)$ and $r(1 - \epsilon/16)$, both of which are always finite by Lemma 4. The proof actually uses these quantities internally, then converts to λ and ρ in order to obtain a more compact final statement.

4.1 THE LEARNING ALGORITHM

Let $\mathcal{A}_{\mathcal{H}}$ be as follows:

1. Input ϵ and δ.

2. $N_0 \leftarrow 32\epsilon^{-1}\delta^{-1}$.

3. $K \leftarrow 0$.

4. $K \leftarrow K + 1$.

5. $N_k \leftarrow K \times N_0$.

6. $C_k \leftarrow \lceil N_k\,(1 - \epsilon/4) \rceil$.

7. $COUNT \leftarrow 0$.

8. $J \leftarrow 0$.

9. $J \leftarrow J + 1$.

10. $X \leftarrow \mathrm{POS}()$.

11. If $|X| > K$ then go to 14.

12. Simulate X for K steps.

13. If X halted then $COUNT \leftarrow COUNT + 1$.

14. If $J < N_k$ then go to 9.

15. If $COUNT < C_k$ then go to 4.

16. Output a program $\mathcal{L}_{\mathcal{H}}$ that does:

    A. Input an instance $\langle \mathcal{M}, \mathcal{I} \rangle$.
    B. Simulate $\mathcal{M}$ on $\mathcal{I}$ for K steps.
    C. If $\mathcal{M}$ has halted then print 1, else print 0.
    D. Halt.

17. Halt.

4.2 PROOF THAT $\mathcal{A}_{\mathcal{H}}$ LEARNS THE HALTING PROBLEM.

Theorem 2 Let $\mathcal{A}_{\mathcal{H}}$ be as stated. Then:

2.A $\mathcal{L}_{\mathcal{H}}$ has an error rate of 0 on negative examples;

2.B with probability greater than $(1 - \delta)$, $\mathcal{L}_{\mathcal{H}}$ has an accuracy greater than $(1-\epsilon)$ on positive examples;

2.C with probability greater than $(1-\delta)$, $\mathcal{A}_{\mathcal{H}}$ executes in time polynomial in δ^{-1}, ϵ^{-1}, λ and ρ;

2.D in the remaining less than δ fraction of cases not bounded in (2.C), $\mathcal{A}_{\mathcal{H}}$ executes in *expected* time polynomial in δ^{-1}, ϵ^{-1}, λ and ρ; and

2.E the probability that $\mathcal{A}_{\mathcal{H}}$ fails to terminate is zero.

Proof:

2.A $\mathcal{L}_{\mathcal{H}}$ has an error rate of 0 on negative examples, as it will classify as positive only those instances which do halt (in K or fewer steps). □ (2.A)

2.B We will use k to refer to the k^{th} iteration of the loop at instructions 4-15, and K to refer to the final value in instruction 16.B.

By definition of r, the learned predicate $\mathcal{L}_{\mathcal{H}}$ has an accuracy on future positive examples greater than $(1-\epsilon)$ if and only if $\mathcal{A}_{\mathcal{H}}$ terminates with $K \geq r(x)$ for some $x > (1 - \epsilon)$. We show that with probability greater than $(1 - \delta)$, $\mathcal{A}_{\mathcal{H}}$ terminates with $K \geq r(1 - \epsilon/2)$.

Let P_k be the probability of halting on the k^{th} iteration. Term a success any instance which runs for more than k steps. $\mathcal{A}_{\mathcal{H}}$ cannot possibly halt on the k^{th} iteration unless there are at most $N_k - C_k = N_k\epsilon/4$ successes. When $k < r(1-\epsilon/2)$ the probability of success is at least $\epsilon/2$, by definition of r. Therefore, by Lemma 2 with $n = N_k = 32k\epsilon^{-1}\delta^{-1}$, $b = 1/2$, and $c = 1/4$, for $k < r(1-\epsilon/2)$ we see P_k is at most $\left[\exp(-2.454\delta^{-1})\right]^k$. By Fact 3, this is less than $(\delta/2.454)^k$.

The probability that $\mathcal{A}_{\mathcal{H}}$ terminates with $K < r(1 - \epsilon/2)$ is less than

$$\sum_{k=1}^{r(1-\epsilon/2)} P_k \left(\prod_{l=1}^{k-1} (1 - P_l) \right)$$

where the first factor is the probability of halting at iteration k and the second is the probability of not having halted before then. Dropping the second factor (which is less than 1) and substituting for P_k shows that this is less than

$$\sum_{k=1}^{r(1-\epsilon/2)} (\delta/2.454)^k$$

By Fact 4 above with $m = 0$ and noting the sum is from $k = 1$, this is less than $(\delta/2.454)/(1 - \delta/2.454)$, which is less than δ for $0 < \delta < 1/2$. □ (2.B)

2.C We show that:

2.C.i with probability at least $1-\delta$, $\mathcal{A}_{\mathcal{H}}$ terminates with K bounded by $16(\lambda + \rho)\epsilon^{-1}$;

2.C.ii the run-time of the k^{th} iteration is bounded by a polynomial in δ^{-1}, ϵ^{-1}, and k;

2.C.iii from which the result follows.

2.C.i For use below (in 2.D,E) we will prove a stronger result, namely that if $k \geq 16(\lambda+\rho)\epsilon^{-1}$ then the probability of terminating on the k^{th} iteration is at least $(1-\delta)$.

On the k^{th} iteration for $k \geq 16(\lambda+\rho)\epsilon^{-1}$, term a success any instance which has length not greater than k, and halts in k or fewer steps. By Lemma 3, $r(1-\epsilon/16) \leq 16\rho\epsilon^{-1}$ and $l(1-\epsilon/16) \leq 16\lambda\epsilon^{-1}$, so the probability of success is at least $(1-\epsilon/8)$. $\mathcal{A}_{\mathcal{H}}$ fails to halt if and only if there are fewer than $C_k = N_k\,(1-\epsilon/4)$ successes. By Lemma 1 with $n = N_k = 32k\epsilon^{-1}\delta^{-1}$, $b = 1/8$, and $c = 1/4$, this probability is at most $\exp(-1.545\delta^{-1}k)$. By Fact 3 this is less than δ. □ (2.C.i)

2.C.ii Instructions 4 through 8, and 15, are executed once on the k^{th} iteration, and are clearly $\mathcal{O}(k\delta^{-1}\epsilon^{-1})$. Instructions 9 through 14 are each executed N_k times on the k^{th} iteration, and except for instruction 12 are clearly also $\mathcal{O}(k\delta^{-1}\epsilon^{-1})$.

Instruction 12 involves simulating the instance returned by POS for k steps. $\mathcal{A}_{\mathcal{H}}$ encodes a UTM to perform this simulation. Because k bounds both the length of the instance and the number of steps simulated, each execution of instruction 12 requires time bounded by some polynomial $Q(k) = \sum_{j=0}^{q} a_j k^j$, where q is the degree of Q and $\{a_j\}$ are its coefficients. Q is constant for $\mathcal{A}_{\mathcal{H}}$, and depends only on the form of UTM employed. In turn Q is bounded by bk^q where $b = \sum_j |a_j|$.

Consequently for some positive constant c, the execution time of each step 9 through 14 is bounded by $ck^q\delta^{-1}\epsilon^{-1}$. These are executed $N_k = 32k\epsilon^{-1}\delta^{-1}$ times on the k^{th} iteration. It follows that, for some positive constant d, the run-time of the k^{th} iteration is bounded by $dk^{q+1}\delta^{-2}\epsilon^{-2}$. □ (2.C.ii)

2.C.iii By (i) and (ii), with probability greater than $(1-\delta)$, the total run-time of $\mathcal{A}_{\mathcal{H}}$ is bounded by

$$d\delta^{-2}\epsilon^{-2} \sum_{k=1}^{16(\lambda+\rho)\epsilon^{-1}} k^{q+1}$$

By Fact 6 with $m = q+1$ this equals

$$d\delta^{-2}\epsilon^{-2} \sum_{j=0}^{q+2} c_j \left(16(\lambda+\rho)\epsilon^{-1}\right)^j$$

where the coefficients c_j depend only on q, which is constant for $\mathcal{A}_{\mathcal{H}}$. □ (2.C.iii)

Consequently, with probability greater than $(1-\delta)$, $\mathcal{A}_{\mathcal{H}}$ executes in time polynomial in δ^{-1}, ϵ^{-1}, ρ and λ. □ (2.C)

2.D The proof of (2.C.iii) showed that the total run-time of $\mathcal{A}_{\mathcal{H}}$ for iterations number 1 through $16(\lambda+\rho)\epsilon^{-1}$ is bounded by a polynomial in δ^{-1}, ϵ^{-1}, ρ and λ. Call this D.

The proof of (2.C.i) showed that if $k \geq 16(\lambda+\rho)\epsilon^{-1}$ then the probability of terminating on the k^{th} iteration is greater than $(1-\delta)$. The proof of (2.C.ii) showed that the run-time of the k^{th} iteration is bounded by $dk^{q+1}\delta^{-2}\epsilon^{-2}$. Consequently, if $\mathcal{A}_{\mathcal{H}}$ runs for more than $16(\lambda+\rho)\epsilon^{-1}$ iterations then its *expected* run-time is bounded by

$$\begin{aligned} &D + \sum_{i=1}^{\infty} d\left(16(\lambda+\rho)\epsilon^{-1}+i\right)^{q+1} \delta^{i-2}\epsilon^{-2} \\ &\quad = D + d\delta^{-2}\epsilon^{-2} \\ &\qquad \times \sum_{j=0}^{q+1} \binom{q+1}{j} \left(16(\lambda+\rho)\epsilon^{-1}\right)^{q-j+1} \sum_{i=1}^{\infty} \delta^i i^j \end{aligned}$$

where we are justified in reversing the order of summation because all terms are non-negative and so the series is absolutely convergent.

Because j is bounded above by $q+1$ this is less than

$$\begin{aligned} &D + d\delta^{-2}\epsilon^{-2} \\ &\quad \times \sum_{j=0}^{q+1} \binom{q+1}{j} \left(16(\lambda+\rho)\epsilon^{-1}\right)^{q-j+1} \sum_{i=1}^{\infty} \delta^i i^{q+1} \end{aligned}$$

By Fact 5 with $m = q+1$ this is less than

$$\begin{aligned} &D + d\delta^{-2}\epsilon^{-2} \\ &\quad \times \sum_{j=0}^{q+1} \binom{q+1}{j} \left(16(\lambda+\rho)\epsilon^{-1}\right)^{q-j+1} 2^{q+2}(q+1)! \end{aligned}$$

which clearly is polynomial in δ^{-1}, ϵ^{-1}, ρ and λ. □ (2.D)

2.E Assume, to the contrary, that the probability with which $\mathcal{A}_{\mathcal{H}}$ fails to halt were some positive number η. The proof of (2.C.i) showed that for $i \geq 0$ the probability that $\mathcal{A}_{\mathcal{H}}$ fails to halt in $i + 16(\lambda+\rho)\epsilon^{-1}$ or fewer iterations is bounded above by δ^i. Consequently for $i > \ln\eta/\ln\delta$ the probability that $\mathcal{A}_{\mathcal{H}}$ fails to halt is less than η, contradicting the assumption that η was that positive number. □ (2.E)

Consequently $\mathcal{A}_{\mathcal{H}}$ learns from examples a total predicate that probably approximates the Halting Problem, executes in time polynomial in δ^{-1}, ϵ^{-1}, ρ and λ with probability greater than $(1-\delta)$ and in *expected* polynomial time otherwise, and always halts. □ (2)

5 IMPORTANT LIMITATIONS.

An important limitation of the proof approach used in this paper is the small size of the hypothesis class (simulation to K) relative to (say) the space of all Boolean functions or all quantified first-order logic formulae. The learners presented in this paper do not have a rich space of learnable knowledge. Although they suffice to prove our main technical result, realistically any learner capable of learning general knowledge must have a richer knowledge representation scheme than appears above.

Another important limitation is the fact that the positive examples are *a priori* known to be generated relative to the Halting Problem. Consequently, the target class has only one element. This fact does not affect the uncomputability of the problems we posed, but it would be an interesting restriction to remove.

Yet another important limitation is the fact that all of our time complexity statements, while polynomial, contain distribution-dependent terms. Another important limitation is the fact that the hypothesis class (simulation to K) fails to be polynomially evaluatable. However, the fact that the underlying uncomputable problem is indeed learnable holds out the hope that other, cleverer, predicates and hypothesis classes may be devised that will satisfy this condition.

In concluding this section, we point out that even if future research were to discover a polynomially evaluatable hypothesis class and a distribution-independent polynomial-time learning algorithm for some uncomputable problem, the test of polynomially learnability would still fail. This is because, by definition, no uncomputable problem can have a polynomially evaluatable target class. One might argue that the definition of polynomially learnable should be changed to drop that requirement on the target class. Alternatively, of course, one might view the learnability of any uncomputable problem to be an undesirable consequence within a theory of learning. This viewpoint might lead one to argue that the definition of learnability should be changed to exclude such results.

Acknowledgements

The paper was much improved by helpful comments and discussion from Gary Borchardt, Clifford Brunk, Pedro Domingos, Christopher Merz, Ron Rivest, Lukas Ruecker, Lynn Stein, Patrick Winston and Zakia Zerhouni. This paper describes research performed at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology sponsored by the National Science Foundation under grant DIR-9121548. Support for the Artificial Intelligence Laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

References

Amsterdam, J. (1988). Some philosophical problems with formal learning theory. *Proc. of the Seventh Natl. Conf. on Artificial Intelligence (AAAI'88),* vol. (2), pp. 580–584.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Control 75,* (2):87–106.

Angluin, D., and Smith, C.H. (1983). Inductive inference: theory and methods. *ACM Computing Surveys 15,* (3):237–269.

Benedek, G.M., and Itai, A. (1991). Learnability with respect to fixed distributions. *Theoretical Computer Sci. 86,* (2):377–389.

Blum, L., and Blum, M. (1975). Toward a mathematical theory of inductive inference. *Information and Control 28,* (2):125–155.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1989). Learnability and the Vapnik–Chervonenkis dimension. *J. of the Assoc. Computing Mach. 36,* (4):929–965.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1987). Occam's razor. *Information Processing Letters 24*, (6):377–380.

Cherniavsky, J.C., and Smith, C.H. (1987). A recursion theoretic approach to program testing. *IEEE Transactions on Software Engineering SE-13,* (7):777–784.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Stat. 23,* 493–509; see P. Erdös and J. Spencer, (1974), *Probabilistic Methods in Combinatorics*, Academic Press, New York, p. 18.

Cutland, N. (1980). *Computability: An Introduction to Recursive Function Theory*, Cambridge University Press, Cambridge, UK.

Ehrenfeucht, A., Haussler, D., Kearns, M., and Valiant, L. (1989). A general lower bound on the number of examples needed for learning. *Information and*

Control 82, (3):247–261.

Ehrenfeucht, A.,and Haussler, D. (1988). Learning decision trees from random examples. *Proc. 1988 Workshop on Computational Machine Learning,* 182–194.

Garasch, W.I., and Smith, C.H. (1992). Learning via queries. *J. of the Assoc. Computing Mach. 39,* (3):649–674.

Gold, E.M. (1967). Language identification in the limit. *Information and Control 10,* (5):447–474.

Herken, R., Ed. (1988). *The Universal Turing Machine: a Half-Century Survey,* Clarendon Press, Oxford University Press, New York.

Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.

Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*, MIT Press, Cambridge, Massachusetts, USA.

Kearns, M. J., and Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *J. of the Assoc. Computing Mach. 41,* (1):67–95.

Lewis, H.R., and Papadimitriou, C.H. (1981). *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, USA, (1981).

Linial, N., Mansour, Y., and Rivest, R. (1991). Results on learnability and the Vapnik–Chervonenkis dimension. *Information and Computation 90,* (1):33–49.

Machlin, R., and Stout, Q.F. (1990). The complex behavior of complex machines. *Physica D 42,* 85–98.

Minsky, M. (1967). *Computation: Finite and Infinite Machines*, (1967),.

Pitt, L. (1990). Special issue on computational learning theory — introduction. *Machine Learning 5,* (2):117–120.

Pitt, L., and Valiant, L.G. (1988). Computational limitations on learning from examples. *J. of the Assoc. Computing Mach. 35,* (4):965–984.

Shvaytser, H. (1990). A necessary condition for learning from positive examples. *Machine Learning 5,* (1):101–113.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. of the London Math. Soc.*, ser. 2, vol. 42:230–265; reprinted in M. Davis, ed., (1965), *The Uncomputable* pp. 115-154, Raven Press, Hewlett, New York.

Valiant, L. G. (1984). A theory of the learnable. *Comm. of the Assoc. Computing Mach. 27,* (11):1134–1142.