

## Equivalence of Models for Polynomial Learnability\*

DAVID HAUSSLER

*Baskin Center for Computer Engineering and Information Sciences,  
University of California at Santa Cruz, Santa Cruz, California 95064*

MICHAEL KEARNS<sup>†</sup>

*Aiken Computation Laboratory, Harvard University,  
Cambridge, Massachusetts 02138*

AND

NICK LITTLESTONE<sup>‡</sup> AND MANFRED K. WARMUTH

*Baskin Center for Computer Engineering and Information Sciences,  
University of California at Santa Cruz, Santa Cruz, California 95064*

In this paper we consider several variants of Valiant's learnability model that have appeared in the literature. We give conditions under which these models are equivalent in terms of the polynomially learnable concept classes they define. These equivalences allow comparisons of most of the existing theorems in Valiant-style learnability and show that several simplifying assumptions on polynomial learning algorithms can be made without loss of generality. We also give a useful reduction of learning problems to the problem of finding consistent hypotheses, and give comparisons and equivalences between Valiant's model and the prediction learning models of Haussler, Littlestone, and Warmuth (*in* "29th Annual IEEE Symposium on Foundations of Computer Science," 1988). © 1991 Academic Press, Inc.

### 1. INTRODUCTION

The model introduced by Valiant (1984) provides the framework for a growing body of research in machine learning (Blumer *et al.*, 1987, 1989; Kearns *et al.*, 1987; Pitt and Valiant, 1988; Rivest, 1987; Valiant, 1984, 1985). This research focuses on understanding the computational complexity of various learning tasks. A central notion is that of polynomial learnability. Roughly speaking, a concept class is said to be polynomially

\* The authors gratefully acknowledge the support of ONR Grant N00014-86-K-0454. M. Kearns was also supported by an AT & T Bell Laboratories Scholarship. This research was done while M. Kearns was visiting the University of California at Santa Cruz.

<sup>†</sup> Current address: AT & T Bell Laboratories, Murray Hill, New Jersey 07974.

<sup>‡</sup> Current address: NEC Research Institute, 4 Independence Way, Princeton, New Jersey 08540.

learnable if there exists an algorithm that can find a hypothesis approximating any concept in the class,<sup>1</sup> when given a polynomial number of examples of the concept and polynomially bounded computational resources. The polynomial growth is with respect to parameters measuring the complexity of the concept, the size of the input to the algorithm, and the accuracy of the resulting approximation.

The specific assumptions and criteria used to define polynomial learnability have varied among the different researchers in the field. To allow confident and accurate comparisons of the results obtained in the different models, it is important to verify the equivalence of the models, or to discover any differences. We perform that task here. The result is a unification of previous work, a precise definition of polynomial learnability, and an understanding of variations in the model that do not affect what is polynomially learnable. A significant part of this paper consists of formal proofs of "folk theorems" that have been known to many researchers in computational learning theory for some time, but have never been documented systematically.

In the first part of this paper (Sections 2 and 3), we consider a number of existing variations of the learning model and show that they lead to equivalent models of polynomial learnability. Some of the equivalence proofs formalize arguments made informally in private communications to the authors; others are new. We show that if all other parameters of the model are equal, then the model where algorithms have access to a single oracle returning labeled examples is equivalent to the model where there are two oracles returning positive and negative examples, respectively; that a model where a learning algorithm must output a good hypothesis with only a fixed probability is equivalent to one where it must do so with arbitrarily high probability; and that without loss of generality, all learning algorithms can be deterministic.

In the process of formalizing the equivalences, we have uncovered interactions between the initial information given to an algorithm and its ability to halt deterministically in all cases. Our results demonstrate that the models used in Blumer *et al.* (1987, 1989), Kearns *et al.* (1987), Pitt and Valiant (1988), Rivest (1987), and Valiant (1984, 1985) are equivalent if probabilistic halting criteria are substituted for deterministic halting in some of the models. The equivalence if deterministic halting is required remains an open question. The results also show the equivalence of other natural variations of these models in nearly all possible combinations of the various modifications that we consider. In only one equivalence proof (the one that shows that the one- and two-oracle models are equivalent) do we

<sup>1</sup>This model has been termed the *PAC-model* by Angluin (1987) standing for *probably approximately correct*.

actually change the distribution on the examples. Thus most of the proven equivalences hold also for learning with respect to any fixed distribution on the examples.

After demonstrating these equivalences in Section 3, we turn to different types of equivalence in Sections 4 and 5. In Blumer *et al.* (1989), a necessary condition for polynomial learnability is given in terms of a combinatorial parameter of the concept class called the Vapnik–Chervonenkis dimension (Vapnik and Chervonenkis, 1971). This condition does not address issues of computational complexity but does address the minimum amount of information needed to perform inductive inference. Using tools from Pitt and Valiant (1988), we show that the problem of polynomially learning  $C$  by  $H$  is equivalent (modulo polynomial-time transformations) to the problem of polynomially finding (with fixed probability) a hypothesis in  $H$  consistent with a given sequence of examples of a target concept in  $C$ , provided that the VC dimension of  $H$  grows only polynomially with the concept complexity measure used for  $C$ . This allows one in many cases to view a learning problem in the more traditional light of complexity-theoretic search problems.

The definitions of learnability discussed here depend fundamentally on the hypothesis space from which the learning algorithm must choose its hypothesis. In many cases it is of interest to study the learnability of a concept class when no restriction is placed on the hypothesis space to be used. For this purpose, a new model of learning was introduced in Haussler *et al.* (1988) that discards the constraints placed on hypotheses of the learning algorithms. In that model the polynomial learning algorithm must predict accurately the label (positive or negative) of an unlabeled random example after receiving polynomially many random examples that are labeled consistently with the target concept. In the main theorem of Section 5 we show that a concept class is polynomially learnable in the prediction model of Haussler *et al.* (1988) iff there exists a polynomially evaluatable (defined in the next section) hypothesis space such that the concept class is polynomially learnable by this hypothesis space.

## 2. MODELS OF POLYNOMIAL LEARNABILITY

### 2.1. Representing Examples and Concepts

**DEFINITION 2.1.** Representation of domains. For each  $n \geq 1$ ,  $X_n$  denotes a set called a *learning domain* on  $n$  attributes. The  $X_n$  for different values of  $n$  are assumed to be disjoint. We let  $\mathbf{X} = \{X_n\}_{n \geq 1}$ . We say a *point* (or *instance*)  $x$  is in  $\mathbf{X}$  if  $x \in \bigcup \{X_n\}_{n \geq 1}$ .

For example,  $X_n$  might be the set of all points in Euclidean  $n$ -dimensional space  $\mathbf{R}^n$  or the Boolean domain  $\{0, 1\}^n$ .<sup>2</sup>

For the purposes of computation, we assume that points are encoded as tuples using any of the standard schemes (see Garey and Johnson, 1979) in such a way that the representation of each point in  $X_n$  has length between  $n$  and  $l = l(n)$ , where  $l(n)$  is a polynomial. The equivalence results given below will not depend on how real numbers are handled (i.e., whether the uniform or logarithmic cost model is used to define the length of representations and inputs to algorithms (Aho *et al.*, 1974). We assume that from the representation of any tuple of  $\mathbf{X}$  one can efficiently determine the unique set  $X_n$  to which it belongs.

We think of concepts as subsets of a learning domain  $X_n$ . For some purposes, we must also have a language in which concepts are represented as strings, and a notion of the size or complexity of a concept, which is usually related to the length of its shortest representation.

**DEFINITION 2.2.** Representations of concepts and concept classes. Given a learning domain  $X$ , a set  $C \subseteq 2^X$  is called a set of *concepts* on  $X$ . A *representation* for  $C$  consists of a set of strings  $L$  and a mapping  $\sigma$  from  $L$  onto  $C$  that associates each string in  $L$  with a concept in  $C$ . A *concept complexity measure* for  $C$  is a mapping **size** from  $C$  to  $\{1, 2, \dots\}$ .

For each  $n \geq 1$ , let  $C_n \subseteq 2^{X_n}$  be a set of concepts,  $L_n$  and  $\sigma_n$  be a representation for  $C_n$ , and **size** <sub>$n$</sub>  be a concept complexity measure for  $C_n$ . Then  $\mathbf{C} = \{(X_n, C_n, L_n, \sigma_n, \mathbf{size}_n)\}_{n \geq 1}$  denotes a *concept class* over  $\mathbf{X}$ . Normally the representation and concept complexity measure will be understood from the context, in which case we will abbreviate  $\mathbf{C}$  as  $\{(X_n, C_n)\}_{n \geq 1}$ . We say that a concept  $c$  is in  $\mathbf{C}$  if  $c \in \bigcup C_n$ . When the concept complexity measure is understood from the context, it is also convenient to let  $C_{n,s}$  denote  $\{c \in C_n : \mathbf{size}(c) \leq s\}$ .

To illustrate these definitions, consider the concept class  $k$ -CNF from Valiant's original paper (1984) for some fixed  $k \geq 1$ . Here  $X_n = \{0, 1\}^n$ , the representation language  $L_n$  consists of all CNF expressions on  $n$  variables (say  $x_1, \dots, x_n$ ) that have at most  $k$  literals per clause,  $C_n$  consists of all  $c \subseteq \{0, 1\}^n$  such that  $c$  is the set of satisfying assignments of one of these expressions,  $\sigma_n$  maps a CNF expression to its set of satisfying assignments, and **size** <sub>$n$</sub> ( $c$ ) is the number of literals in the smallest  $k$ -CNF representation of  $c$ . The concept classes  $k$ -DNF,  $k$ -term DNF, and  $k$ -clause CNF are defined similarly by restricting to DNF expressions with at most  $k$  literals per term, DNF expressions with at most  $k$  terms, and CNF expressions with at most  $k$  clauses, respectively. The concept classes DNF and CNF

<sup>2</sup> We assume that any learning domain  $X_n$  that we consider can be embedded in  $\mathbf{R}^n$  by a measurable embedding.

can also be defined in this manner. Here  $C_n = 2^{\{0,1\}^n}$ , but  $C_{n,s}$  includes only those concepts that can be represented by a DNF (resp. CNF) expression with at most  $s$  literals.

For real-valued domains, examples of concept classes include the class of closed halfspaces and the class of closed convex polytopes. In the case of halfspaces in  $n$  dimensions, one possibility is to assume that concepts are represented by the  $n + 1$  coefficients of the separating hyperplane, and that  $\text{size}_n(c) = n + 1$  for all  $c \in C_n$  (uniform cost model). In the case of convex polytopes, concepts might be represented either as the intersection of a set of halfspaces, or by specifying the vertices of the convex polytope.

**DEFINITION 2.3.** Examples and hypotheses. Let  $\mathbf{C}$  be a concept class over  $\mathbf{X}$ . Given a concept  $c$  in  $\mathbf{C}$ , we define an *example* of  $c$  to be a pair  $\langle x, a \rangle$ , where  $x$  is in  $\mathbf{X}$  and  $a \in \{0, 1\}$  such that  $a = 1$  iff  $x \in c$ . If  $x \in c$  then  $\langle x, 1 \rangle$  is called a *positive* example of  $c$ , and if  $x \notin c$  then  $\langle x, 0 \rangle$  is called a *negative* example of  $c$ . A *sample* of  $c$  is a sequence of examples of  $c$ . As above, for computational purposes we assume samples are represented as sequences of pairs using any of the standard schemes. The *size* of a sample is the number of examples it contains. The *length* of a sample is the number of symbols in its encoding. (In case of the uniform cost model, each real number contributes one to the length.)

Let  $\mathbf{C}$  and  $\mathbf{H}$  be two concept classes. An algorithm for learning  $\mathbf{C}$  by  $\mathbf{H}$  is an algorithm that when given examples of some concept  $c \in \mathbf{C}$  will produce as output (a representation of) some concept  $h \in \mathbf{H}$  that is an approximation of  $c$  in a sense made precise below. The class  $\mathbf{C}$  is called the *target class* and  $c$  is called the *target concept*. The class  $\mathbf{H}$  is called the *hypothesis space* used by the algorithm and the hypothesis  $h$  (whose representation is output by the algorithm) is called the *hypothesis* of the algorithm.

We say that a hypothesis  $h$  in  $\mathbf{H}$  is *consistent* with a sample  $\langle x_1, a_1 \rangle, \dots, \langle x_m, a_m \rangle$  of  $c$  if  $x_i \in h \Leftrightarrow a_i = 1$  for all  $1 \leq i \leq m$ .

## 2.2. Models for Polynomial Learnability

We now define the three most popular variants of Valiant's original model. In the following models, we assume that  $\mathbf{X}$  is a learning domain and that  $\mathbf{C}$  and  $\mathbf{H}$  are concept classes over  $\mathbf{X}$ .

*Model 1. The Functional Model.* In this model a learning algorithm implements a function that maps from samples to hypotheses. If  $c \in C_n$  is a given target concept,  $h \in H_n$  is a hypothesis, and  $D$  is any fixed probability distribution on  $X_n$ , then define the *error* of  $h$  (with respect to  $c$  and  $D$ ) to be the probability that  $h$  is inconsistent with a random example of  $c$  (i.e., an example  $\langle x, a \rangle$  of  $c$  in which  $x$  is drawn randomly

from  $X_n$  according to  $D$ ). We will say that  $\mathbf{C}$  is *polynomially learnable by  $\mathbf{H}$  (in the functional model)* if there exists an algorithm  $A$  that takes as input a sample of a target concept in  $\mathbf{C}$ , and outputs a representation of a hypothesis in  $\mathbf{H}$  such that the following property holds:

*Property 1.* (a) There is a function  $m(\varepsilon, \delta, n, s)$ , polynomial in  $1/\varepsilon$ ,  $1/\delta$ ,  $n$  and  $s$ , such that for all  $0 < \varepsilon$ ,  $\delta < 1$ , and  $n, s \geq 1$ , and for all target concepts  $c \in C_{n,s}$  and all probability distributions  $D$  on  $X_n$ , if  $A$  is given a random sample of  $c$  of at least  $m(\varepsilon, \delta, n, s)$  examples drawn independently according to  $D$ , then  $A$  produces a representation in  $\mathbf{H}$  of a hypothesis  $h \in H_n$ , and with probability at least  $1 - \delta$  the hypothesis  $h$  has error at most  $\varepsilon$ .

(b) Algorithm  $A$  runs in time polynomial in the length of its input.

We let  $S_A(\varepsilon, \delta, n, s)$  denote the smallest sample size  $m(\varepsilon, \delta, n, s)$  such that Property 1(a) holds for algorithm  $A$ .  $S_A(\varepsilon, \delta, n, s)$  is called the *sample complexity* of  $A$ . Note that in the functional model the algorithm  $A$  is not given any of the parameters  $\varepsilon$ ,  $\delta$ ,  $n$ , and  $s$  as input. The only input to  $A$  is a batch of examples. Throughout the paper  $\varepsilon$  will be called the *accuracy parameter* and  $\delta$  the *confidence parameter*.

*Model 2. The One-Oracle Model.* Instead of specifying that  $A$  be simply a function mapping samples to hypotheses, we can allow  $A$  to explicitly use information about the desired accuracy and confidence parameters  $\varepsilon$  and  $\delta$ , as well as the complexity parameters  $n$  and  $s$ . This can be accomplished by giving  $\varepsilon$ ,  $\delta$ ,  $n$ , and  $s$  as input to  $A$  and supplying  $A$  with an oracle  $EX$  for random examples of the target concept. Each time  $EX$  is called, it selects an instance in  $X_n$  independently at random according to the distribution  $D$  and returns it along with a label indicating whether or not it is in the target concept. Throughout this paper whenever an oracle returns an example to an algorithm, then we charge the algorithm with time equal to the length of the received example. We say that  $\mathbf{C}$  is *polynomially learnable by  $\mathbf{H}$  (in the one-oracle model)* if there is an algorithm  $A$  taking inputs  $\varepsilon$ ,  $\delta$ ,  $n$ , and  $s$  and outputting a representation of a hypothesis in  $\mathbf{H}$  such that the following property holds:

*Property 2.* For all  $0 < \varepsilon$ ,  $\delta < 1$ , and  $n, s \geq 1$ , and for all target concepts  $c \in C_{n,s}$  and all probability distributions  $D$  on  $X_n$ ,

(a)  $A$  outputs a representation in  $\mathbf{H}$  of a hypothesis  $h \in H_n$ , and with probability at least  $1 - \delta$  the output hypothesis  $h$  has error at most  $\varepsilon$ .

(b) The total running time is bounded by a polynomial in  $1/\varepsilon$ ,  $1/\delta$ ,  $n$ , and  $s$ .

In this model, the sample complexity  $S_A(\varepsilon, \delta, n, s)$  of an algorithm  $A$  is taken to be the worst-case number of oracle calls on input  $\varepsilon$ ,  $\delta$ ,  $n$ ,  $s$ , over

all  $c \in C_{n,s}$  and all sequences of examples of  $c$ . Similarly, the worst-case running time is denoted  $T_A(\varepsilon, \delta, n, s)$ .

*Model 3. The Two-Oracle Model.* The one-oracle model can be further modified to allow the algorithm  $A$  access to two oracles, one that returns random positive examples of the target concept (which we will call *POS*), and one that returns random negative examples (which we will call *NEG*). In this case there are two distributions  $D^+$  and  $D^-$ .  $D^+$  is a distribution on  $c$  and  $D^-$  is a distribution on  $X_n - c$ . Calls to *POS* return examples chosen according to  $D^+$  and calls to *NEG* return examples chosen according to  $D^-$ .

In this two-oracle model we also define two types of error: *error*<sup>+</sup> (the *positive error*) is the probability that a random example from *POS* is classified as negative by the hypothesis, and *error*<sup>-</sup> (the *negative error*) is the probability that a random example from *NEG* is classified as positive by the hypothesis. The definition of polynomial learnability is now as in the one-oracle model, except that in Property 2 we now require the hypothesis of learning algorithm  $A$  to have both positive error at most  $\varepsilon$  and negative error at most  $\varepsilon$ , that is:

*Property 3.* For all  $0 < \varepsilon, \delta < 1$  and  $n, s, \geq 1$ , and for all target concepts  $c \in C_{n,s}$  and all probability distributions  $D$  on  $X_n$ ,

(a)  $A$  outputs a representation in  $\mathbf{H}$  of a hypothesis  $h \in H_n$ , and with probability at least  $1 - \delta$  the output hypothesis  $h$  has positive error at most  $\varepsilon$  and negative error at most  $\varepsilon$ .

(b) The total running time is bounded by a polynomial in  $1/\varepsilon, 1/\delta, n$  and  $s$ .

The main contribution of this paper is to prove the equivalence of these models, along with a number of additional variations. The equivalence results depend only on weak assumptions about the target class and the hypothesis space, outlined below. The variations that we consider are:

(1) *Randomized vs. deterministic:* We consider both randomized and deterministic learning algorithms. Randomized algorithms are allowed to make use of flips of a fair coin. These coin flips are independent of the random examples of the target concept received by the algorithm. A randomized algorithm is charged one unit of time for each coin-flip that it uses. The sample complexity  $S_A(\varepsilon, \delta, n, s)$  and the running time  $T_A(\varepsilon, \delta, n, s)$  are extended to worst case measures over the coin-flips of the algorithm. We show that without loss of generality, with respect to polynomial learnability all learning algorithms are deterministic (modulo some weak regularity assumptions on the hypothesis space that we discuss below).

(2) Polynomial in  $1/\delta$  vs. polynomial in  $\log(1/\delta)$  vs. fixed  $\delta$ : We consider three ways to treat the confidence parameter  $\delta$ . In one case the sample and time complexities of a polynomial learning algorithm are required to be polynomial in  $1/\delta$ , as described above. In the second case, the sample and time complexities are required to be polynomial in  $\log 1/\delta$ . In the final case we fix  $\delta = \delta_0$  for any constant  $0 < \delta_0 < 1$  (so the learning algorithm no longer requires  $\delta$  as an input), and we require only that a polynomial learning algorithm achieve this level of confidence, with sample and time complexity polynomial in the remaining variables of the model. We show that in all three cases, the same class of polynomially learnable concept classes is obtained.

For the one-oracle and two-oracle models, we also consider the case in which the value of  $s$  is not given as input to the algorithm. We will show that such a model is equivalent to the model where  $s$  is given, provided that one also relaxes the halting criterion, changing it to a probabilistic halting criterion. Similar results are obtained in Linial *et al.* (1988) (see also Benedek and Itai (1988b)). We thus consider the following two further modifications:

(3) Knowledge of  $s$  vs. no knowledge of  $s$ .

(4) Deterministic halting vs. probabilistic halting. For the probabilistic halting case, property 2 in the definitions of the one-oracle model is changed to:

*Property 2'*. There exists some polynomial  $p$  such that for all  $0 < \varepsilon$ ,  $\delta < 1$ , and  $n, s \geq 1$ , and for all target concepts  $c \in C_{n,s}$  and all probability distributions  $D$  on  $X_n$ , with probability at least  $1 - \delta$ ,

(a)  $A$  outputs a representation in  $\mathbf{H}$  of a hypothesis in  $H_n$  that has error at most  $\varepsilon$ .

(b) The total running time is bounded by  $p(1/\varepsilon, 1/\delta, n, s)$ .

Property 3 in the definition of the two-oracle model is changed analogously. Sample complexity is not defined in the probabilistic halting case. Note that Property 2' allows for the possibility that the expected running time of the learning algorithm is infinite.

For convenience of notation, we will introduce a parameterized version of the three basic models along with their various modifications. Thus, **functional**( $p_1, p_2$ ) will denote those pairs of concept classes  $(\mathbf{C}, \mathbf{H})$  such that  $\mathbf{C}$  is polynomially learnable by  $\mathbf{H}$  in the functional model under modifications  $p_1$  and  $p_2$ , where  $p_1$  is either  $1/\delta$  (when polynomial dependence on  $1/\delta$  allowed),  $\log(1/\delta)$  (for restriction to polynomial dependence on  $\log(1/\delta)$ ), or *fixed*  $\delta$  (when we fix  $\delta$  to  $0 < \delta_0 < 1$ ), and  $p_2$  is either *rand* (for randomized learning algorithms) or *det* (for deterministic algo-

rithms). Similarly, **one-oracle**( $p_1, p_2, p_3, p_4$ ) and **two-oracle**( $p_1, p_2, p_3, p_4$ ) will denote those pairs of concept classes  $(C, H)$  such that  $C$  is polynomially learnable by  $H$  in the one- and two-oracle models respectively, under modifications  $p_1, p_2, p_3$  and  $p_4$ . Here  $p_1$  and  $p_2$  are as above, and  $p_3$  is either *s-known* or *s-unknown* (according to whether  $s$  is given to the learning algorithm or not) and  $p_4$  is either *always-halts* or *usually-halts* (according to whether learning algorithms are required to halt or not).

As examples, **functional**( $\log(1/\delta), rand$ ) is the set of all pairs of concept classes  $(C, H)$  such that  $C$  is polynomially learnable by  $H$  in the functional model by randomized algorithms with sample complexity polynomially dependent on  $\log(1/\delta)$ , and **two-oracle**( $1/\delta, det, s-unknown, usually-halts$ ) is the set of all pairs of concept classes  $(C, H)$  such that  $C$  is polynomially learnable by  $H$  in the two-oracle model by deterministic algorithms with running time polynomial in  $1/\delta$  and no explicit knowledge of  $s$  that halt probabilistically.

In the following section we demonstrate that all three of the basic models are equivalent to each other, and that models resulting from all combinations of the above variations are equivalent to each other, except for the restriction mentioned above that if an oracle algorithm does not know  $s$  then the halting criterion of the learning algorithm is probabilistic.

In all the models described above the algorithm has to perform well for all probability distributions. For each model one can define learnability with respect to a fixed distribution (or two fixed distributions in the two-oracle case) (Benedek and Itai, 1988a). The question arises which of the equivalences proven in this paper still hold if learnability is defined with respect to fixed distributions. Interestingly enough all equivalences hold in that case as well, with the exception of the equivalence between the one- and two-oracle models.

We make some general assumptions about the concept classes  $C$  and  $H$  that hold throughout the following analysis. First we assume that the language used for representing hypotheses in  $H$  is such that one can efficiently determine if an instance is a member of a given hypothesis. Formally, we assume that there is a polynomial algorithm that, given a string  $w$  in  $L_n$  and a representation of an instance  $x \in X_n$ , determines whether or not  $x \in \sigma_n(w)$ . Such an  $H$  is called *polynomially evaluable*. Second, when the domain is real-valued we also assume that all concepts are Borel sets, and that all sets of concepts are well-behaved in the measure-theoretic sense defined in Blumer *et al.* (1989). We let **regular**<sub>1</sub> denote the set of all pairs  $(C, H)$  that satisfy the above regularity assumptions.

For certain of the relationships among the models we require stronger regularity assumptions. We let **regular**<sub>2</sub> be the set of all pairs  $(C, H)$  in **regular**<sub>1</sub> such that for each  $H_n \in H$  we have  $\emptyset \in H_n, X_n \in H_n$ , and for all  $x \in X_n, \{x\} \in H_n$ .

### 3. Equivalence Results of Learnability Models

In this section we prove our main theorem which shows the equivalence of the learnability models introduced in the previous section. As the only restriction we require that if  $s$  *unknown*, then the halting criterion must be *usually-halts*.

**THEOREM 3.1.** *If  $(\mathbf{C}, \mathbf{H}) \in \mathbf{regular}_2$  is an element of any of the following sets, then it is an element of all of them:*

**functional**( $p_1, p_2$ ),  
**one-oracle**( $p_1, p_2, p_3, p_4$ ),  
**two-oracle**( $p_1, p_2, p_3, p_4$ ).

Here

$p_1 \in \{\log 1/\delta, 1/\delta, \text{fixed } \delta\}$ ,  
 $p_2 \in \{\text{rand}, \text{det}\}$ ,  
 $p_3 \in \{s\text{-known}, s\text{-unknown}\}$ ,  
*if  $p_3 = s\text{-known}$  then  $p_4 \in \{\text{always-halts}, \text{usually-halts}\}$ ,*  
*if  $p_3 = s\text{-unknown}$  then  $p_4 = \text{usually-halts}$ .*

Figure 1 presents a graph whose vertices represent the models that we show to be equivalent in this theorem. The directed edges represent the implications that we will directly demonstrate in our proof of the theorem. Some of these edges are labeled with the numbers of the lemmas in which the corresponding implications are demonstrated. The other implications are considered below.

*Proof of Theorem 3.1.* The implications corresponding to the unlabeled edges of Fig. 1 follow immediately from the following observations. A deterministic algorithm for learning  $\mathbf{C}$  by  $\mathbf{H}$  in some model is also a randomized algorithm for learning  $\mathbf{C}$  by  $\mathbf{H}$  in the model that differs only in replacing the parameter *det* with *rand*. Similarly, a deterministically halting algorithm is also a probabilistically halting algorithm. A learning algorithm for a model in which  $s$  is not available to the algorithm can also be used when  $s$  is available: it just ignores  $s$ . The  $1/\delta$  models have been omitted from the diagram, but would fit in the middle of each of the downward pointing arrows between the  $\log 1/\delta$  and fixed- $\delta$  models. For any particular set of values of the other parameters, learnability in the  $\log 1/\delta$  model implies learnability in the corresponding  $1/\delta$  model, and learnability in the  $1/\delta$  model implies learnability in the corresponding fixed- $\delta$  model. To see this note that an algorithm that is polynomial in  $\log 1/\delta$  is polynomial in  $1/\delta$ . An algorithm in either of the oracle models that is polynomial in  $1/\delta$

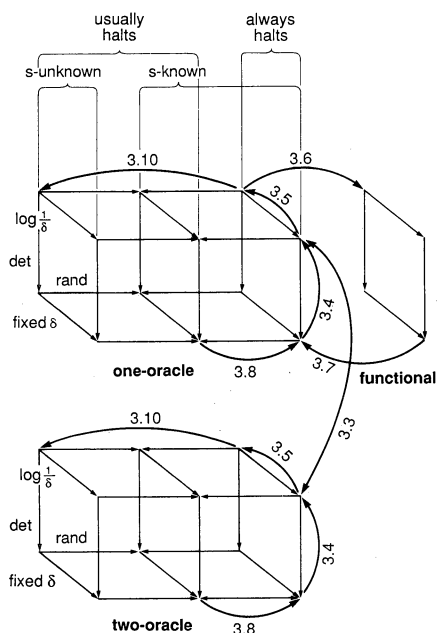


FIG. 1. Equivalences among models of learnability. The numbers on the arrows indicate the corresponding lemmas. The top, front, bottom and backside of the cubes are labeled with  $\log \frac{1}{\delta}$ , rand, fixed  $\delta$ , and det, respectively.

(and the other relevant parameters) can also be used as a polynomial algorithm in the fixed- $\delta$  model: just set  $\delta$  to  $\frac{1}{2}$  in its input. A functional algorithm does not receive  $\delta$  as input, but again, an algorithm for a  $1/\delta$  model will also work in the corresponding fixed- $\delta$  model. These observations cover the cases corresponding to the unlabeled arrows in the figure. The implications corresponding to the labeled arrows are demonstrated in the indicated lemmas. The proof of the theorem is completed by the following lemma, which follows by inspection in Fig. 1. ■

LEMMA 3.1. *The directed graph given in Fig. 1 is strongly connected.*

In the results of the rest of the paper we will make use of the following bounds on the tails of the binomial distribution (Angluin and Valiant, 1979).

LEMMA 3.2 (Chernoff Bounds). *For  $0 \leq p \leq 1$  and  $m$  a positive integer, let  $LE(p, m, r)$  denote the probability of at most  $r$  successes in  $m$  independent trials of a Bernoulli variable with probability of success  $p$ , and let  $GE(p, m, r)$  denote the probability of at least  $r$  successes. Then if  $0 \leq \alpha \leq 1$ ,*

*Fact 1.  $LE(p, m, (1 - \alpha)mp) \leq e^{-\alpha^2 mp/2}$  and*

*Fact 2.*  $GE(p, m, (1 + \alpha)mp) \leq e^{-\alpha^2 mp/3}$ .

LEMMA 3.3.

$$\begin{aligned} & \mathbf{one-oracle} \left( \log \frac{1}{\delta}, \mathit{rand}, \mathit{s-known}, \mathit{always-halts} \right) \cap \mathbf{regular}_2 \\ &= \mathbf{two-oracle} \left( \log \frac{1}{\delta}, \mathit{rand}, \mathit{s-known}, \mathit{always-halts} \right) \cap \mathbf{regular}_2. \end{aligned}$$

*Proof.* We first show that polynomial learnability in the one-oracle model implies polynomial learnability in the two-oracle model. Let  $(\mathbf{C}, \mathbf{H}) \in \mathbf{one-oracle}(\log 1/\delta, \mathit{rand}, \mathit{s-known}, \mathit{always-halts}) \cap \mathbf{regular}_2$ , and let  $A_1$  be an algorithm learning  $\mathbf{C}$  by  $\mathbf{H}$  in this model. We construct an algorithm  $A_2$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  in the model  $\mathbf{two-oracle}(\log 1/\delta, \mathit{rand}, \mathit{s-known}, \mathit{always-halts})$ .

Let  $c \in \mathbf{C}$  be the target concept over domain  $X$  to be learned in the two-oracle model, and let  $D^+$  and  $D^-$  be the probability distributions over the positive and negative examples of  $c$ , respectively. We now define a single distribution  $D$  over the entire domain  $X$  by

$$D(x) = \frac{1}{2}D^+(x) + \frac{1}{2}D^-(x)$$

for each  $x \in X$ . We now use  $A_1$  to learn  $c$  as follows: on inputs  $\varepsilon, \delta, n$ , and  $s$ , algorithm  $A_2$  simulates algorithm  $A_1$  with inputs  $\varepsilon/2, \delta, n$ , and  $s$ . Each time algorithm  $A_1$  requests an example from the oracle  $EX$ , algorithm  $A_2$  flips a fair coin. If the outcome is heads,  $A_2$  calls  $POS$  and gives the positive example returned by the oracle to  $A_1$ , along with a label indicating that this example is positive. If the outcome is tails,  $A_2$  calls  $NEG$  and gives the returned negative example to  $A_1$ , along with a label indicating that the example is negative. Thus the examples given to  $A_1$  on calls to  $EX$  are drawn independently from the distribution  $D$  defined above.

Suppose that  $h$  is the hypothesis output by algorithm  $A_1$  following this simulation, and let  $e, e^+$ , and  $e^-$  denote the error of  $h$  on  $D, D^+$ , and  $D^-$ , respectively. Since  $A_1$  must work for any distribution, we must have that  $e \leq \varepsilon/2$  with probability at least  $1 - \delta$ . But

$$e = \frac{1}{2}e^+ + \frac{1}{2}e^-$$

so with probability at least  $1 - \delta$ ,

$$\frac{1}{2}e^+ \leq \frac{\varepsilon}{2} \quad \text{and} \quad \frac{1}{2}e^- \leq \frac{\varepsilon}{2},$$

that is, as desired both  $e^+$  and  $e^-$  are at most  $\varepsilon$  with probability at least  $1 - \delta$ .

For the other direction, let  $(C, H) \in \text{two-oracle}(\log 1/\delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_2$  and let  $A_2$  be an algorithm learning  $C$  by  $H$  in this model. Now we construct an algorithm  $A_1$  for learning  $C$  by  $H$  in the model  $\text{one-oracle}(\log 1/\delta, \text{rand}, s\text{-known}, \text{always-halts})$ .

Let  $c \in C$  be the target concept and  $D$  be the distribution over  $X$ . Let  $m = S_{A_2}(\varepsilon, \delta/3, n, s)$  be the sample complexity of  $A_2$  when called with the parameters  $\varepsilon, \delta/3, n$ , and  $s$ , and let  $p^+$  (respectively,  $p^-$ ) denote the probability of drawing a positive (respectively, negative) example of  $c$  from the distribution  $D$ . If  $p^+ \geq \varepsilon$  (respectively  $p^- \geq \varepsilon$ ), then the probability of getting at most  $m$  positive (respectively negative) examples of  $c$  in  $q$  calls to  $EX$  is at most  $LE(\varepsilon, q, m)$ . So choose  $q \geq (2/\varepsilon)m$ , and apply Lemma 3.2 with  $m' = (2/\varepsilon)m$  and  $\alpha = 1/2$  to get

$$LE(\varepsilon, q, m) \leq LE\left(\varepsilon, \frac{2}{\varepsilon}m, m\right) = LE\left(\varepsilon, m', \left(1 - \frac{1}{2}\right)m'\varepsilon\right) \leq e^{-m'/4}.$$

Solving  $e^{-m'/4} \leq \delta/3$  gives  $m \geq 4 \ln(3/\delta)$  and  $q \geq (8/\varepsilon) \ln(3/\delta)$ . Thus, if  $p^+ \geq \varepsilon$ , the probability that we fail to draw at least  $m$  positive examples in  $\max((2/\varepsilon)m, (8/\varepsilon) \ln(3/\delta))$  calls to  $EX$  is at most  $\delta/3$ . The same fact holds for negative examples and  $p^-$ .

Algorithm  $A_1$  uses this fact as follows:  $A_1$  sets  $\varepsilon$  to  $\frac{1}{2}$  if the input value for  $\varepsilon$  was greater than  $\frac{1}{2}$ ; otherwise it leaves  $\varepsilon$  alone. It then makes  $\max((2/\varepsilon)m, (8/\varepsilon) \ln(3/8))$  calls to the oracle  $EX$ . If  $A_1$  fails to draw at least  $m$  positive (respectively,  $m$  negative) points, the hypothesis  $\emptyset$  (respectively,  $X$ ) is output. If both  $m$  positive and  $m$  negative examples were obtained, then  $A_1$  simulates  $A_2$  on the input parameters  $\varepsilon, \delta/3, n$ , and  $s$ , using the positive and negative examples received from  $EX$  to answer  $A_2$ 's calls to  $POS$  and  $NEG$ .

Let  $h$  be the hypothesis output by  $A_2$  following this simulation. Define the conditional distributions  $D^+(x) = D(x)/p^+$  if  $x \in c$ ,  $D^+(x) = 0$  otherwise, and  $D^-(x) = D(x)/p^-$  if  $x \notin c$ ,  $D^-(x) = 0$  otherwise. Let  $e$  denote the error of  $h$  on distribution  $D$  and  $e^+$  and  $e^-$  denote the error of  $h$  on the conditional distributions  $D^+$  and  $D^-$ , respectively.

If both  $p^+$  and  $p^-$  are at least  $\varepsilon$  then with probability at least  $1 - 2\delta/3$ , the algorithm will receive at least  $m$  positive and at least  $m$  negative examples. If this occurs then  $A_1$  simulates  $A_2$  with parameters  $\varepsilon, \delta/3$ , and  $s$ . Then with probability at least  $1 - \delta/3$ ,  $A_2$  will return a hypothesis  $h$  for which  $e^+ \leq \varepsilon$  and  $e^- \leq \varepsilon$ . If  $h$  is such a hypothesis, then the error of  $h$  with respect to  $D$  is  $e = p^+e^+ + p^-e^- = p^+e^+ + (1 - p^+)e^- \leq \varepsilon$ . Thus when both  $p^+$  and  $p^-$  are at least  $\varepsilon$  the probability of finding a hypothesis with error no more than  $\varepsilon$  is at least  $1 - \delta$ . If  $p^+ < \varepsilon$  and  $p^- \geq \varepsilon$  then with probability at least  $1 - \delta/3$  at least  $m$  negative examples will be drawn. In this case either the hypothesis  $\emptyset$  will be output or  $A_2$  will be simulated. The hypothesis  $\emptyset$

has error  $p^+ < \varepsilon$ . If  $A_2$  is simulated, then with probability at least  $1 - \delta/3$  it will produce a hypothesis with error at most  $\varepsilon$ . Thus when  $p^+ < \varepsilon$  and  $p^- \geq \varepsilon$  a hypothesis with error at most  $\varepsilon$  will be produced with probability at least  $1 - 2\delta/3$ . The case where  $p^- < \varepsilon$  and  $p^+ \geq \varepsilon$  is similar to this case. The final case where both  $p^+$  and  $p^-$  are less than  $\varepsilon$  cannot occur since  $\varepsilon \leq \frac{1}{2}$ . ■

LEMMA 3.4.

$$\begin{aligned} & \text{one-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1 \\ & \subseteq \text{one-oracle}\left(\log \frac{1}{\delta}, \text{rand}, s\text{-known}, \text{always-halts}\right) \cap \text{regular}_1 \end{aligned}$$

and

$$\begin{aligned} & \text{two-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1 \\ & \subseteq \text{two-oracle}\left(\log \frac{1}{\delta}, \text{rand}, s\text{-known}, \text{always-halts}\right) \cap \text{regular}_1. \end{aligned}$$

*Proof.* The key idea is to run the algorithm for fixed  $\delta$  many times until it has produced a good hypothesis with high probability. We then test the hypotheses this algorithm has produced to find the good one. We give the proof for the one-oracle model. The proof for the two-oracle model is similar. Let  $(\mathbf{C}, \mathbf{H}) \in \text{one-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1$  and let  $A$  be an algorithm learning  $\mathbf{C}$  by  $\mathbf{H}$  in this model when  $\delta$  is fixed at some  $0 < \delta_0 < 1$ . We construct an algorithm  $B$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  in the model  $\text{one-oracle}(\log(1/\delta), \text{rand}, s\text{-known}, \text{always-halts})$ .

On inputs  $\varepsilon, \delta, n$  and  $s$ , algorithm  $B$  simulates algorithm  $A$  on inputs  $\varepsilon/4, n$ , and  $s$  for  $k = \lceil \log_{(1/\delta_0)}(3/\delta) \rceil$  times. Since  $A$  always halts, these  $k$  runs yield  $k$  hypotheses  $h_1, h_2, \dots, h_k$  in  $\mathbf{H}$ . Now  $B$  draws  $(12/\varepsilon) \ln(3k/\delta)$  additional examples and outputs a hypothesis that agrees with the largest number of these examples.

It is easy to see that since the sample and time complexities of  $A$  are polynomial in  $\varepsilon, n$ , and  $s$ , the sample and time complexities of  $B$  are polynomial in  $\varepsilon, \log 1/\delta, n$ , and  $s$ . Thus we only have to show that the hypothesis output by  $B$  has error at most  $\varepsilon$  with probability at least  $1 - \delta$ .

First, observe that the probability that there is no  $h_i$  with error at most  $\varepsilon/4$  is at most  $\delta_0^k \leq \delta/3$ . If a hypothesis has error greater than  $\varepsilon$ , then the probability that it agrees with at least a fraction  $1 - \varepsilon/2$  of a sample of size  $m$  is bounded above by

$$LE(\varepsilon, m, (\varepsilon/2)m) \leq e^{-(m\varepsilon/8)}$$

Thus second, the probability that some  $h_i$  with error greater than  $\varepsilon$  agrees with at least a fraction  $1 - (\varepsilon/2)$  is at most  $ke^{-m\varepsilon/8} \leq \delta/3$  for  $m \geq 8/\varepsilon \ln(3k/\delta)$ . On the other hand, the probability that a hypothesis with error at most  $\varepsilon/4$  agrees with less than a fraction  $1 - (\varepsilon/2)$  of a sample of size  $m$  is bounded above by

$$GE(\varepsilon/4, m, (\varepsilon/2)m) \leq e^{-m\varepsilon/12}.$$

Thus, third, the probability that some  $h_i$  with error at most  $\varepsilon/4$  agrees with less than a fraction  $1 - (\varepsilon/2)$  is at most  $ke^{-m\varepsilon/12} \leq \delta/3$  for  $m \geq (12/\varepsilon) \ln(2k/\delta)$ .

The probability that  $B$  fails to find a hypothesis with error at most  $\varepsilon$  is at most the probability that no  $h_i$  has error less than  $\varepsilon/4$ , plus the probability that some  $h_i$  with error greater than  $\varepsilon$  agrees with at least a fraction  $1 - (\varepsilon/2)$  of the  $m$  examples used for the hypothesis test, plus the probability that an  $h_i$  with error less than  $\varepsilon/4$  disagrees with more than a fraction  $\varepsilon/2$  of the  $m$  examples. Since  $B$  draw at least  $(12/\varepsilon) \ln(3k/\delta)$  additional examples, the sum of these probabilities is bounded by  $\delta/3 + \delta/3 + \delta/3 = \delta$  (above three cases). ■

LEMMA 3.5.

$$\begin{aligned} & \text{one-oracle}(\log 1/\delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_2 \\ & \subseteq \text{one-oracle}(\log 1/\delta, \text{det}, s\text{-known}, \text{always-halts}) \cap \text{regular}_2 \end{aligned}$$

and

$$\begin{aligned} & \text{two-oracle}(\log 1/\delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_2 \\ & \subseteq \text{two-oracle}(\log 1/\delta, \text{det}, s\text{-known}, \text{always-halts}) \cap \text{regular}_2. \end{aligned}$$

*Proof.* The key idea of the proof is to use the randomness of the order of the examples produced by an oracle to simulate a fair coin. We first describe how this is done. We distinguish two cases. In this first case we are in the one-oracle model and randomness is extracted from the single oracle. In the second, the two-oracle model, we show how to extract randomness from *POS*.

*Case 1.* The label of an example from the oracle constitutes a biased coin. We use a trick of von Neumann to convert this coin into a fair coin. Draw a pair of examples and consider the order of their labels. The sequences  $\langle +, - \rangle$  and  $\langle -, + \rangle$  are equally likely and thus constitute the two events of a fair coin. In the case when the labels are equal the pair is discarded and a new pair is drawn. Let  $p$  denote the probability that the labels of a pair are different and let  $q$  be the probability of the most likely

label when drawing one example. Observe that  $p \geq 1 - q$ . Note also that if  $p$  is very small then too many pairs need to be drawn to obtain a fair coin flip. In that case we stop with the default hypothesis ( $\emptyset$  if the most frequent label in a large enough sample is  $-$  and the whole domain if the most frequent label in the sample is  $+$ ).

*Case 2.* In the two-oracle model the learning algorithm decides whether to draw the next example from *POS* or from *NEG*. Thus the label of the example cannot be used as a source of randomness. We show a method of extracting randomness from the examples of *POS*. (In a similar manner, one can extract randomness from *NEG* or, in the one-oracle case, from the examples drawn from the single oracle without using the labels of the examples.) We first establish an arbitrary linear order on the domain. In multidimensional spaces we can use a simple lexicographic ordering. Then we repeatedly draw pairs of points from *POS*. If the points of the current pair are the same, they are discarded. However, if they are different, then the larger point is just as likely to occur in the first position as it is in the second. As above, let  $p$  be the probability that the points of a pair are different and let  $q$  be the probability of the most likely point. Again  $p \geq 1 - q$  and if  $p$  is very small then we stop with a default hypothesis. Here the default hypothesis is  $\{x\}$ , where  $x$  is the point that occurs with highest frequency in a large enough sample. Note that the intersection with **regular**<sub>2</sub> assures that in both cases the default hypotheses are in the hypothesis class.

In both the one- and two-oracle models we have described how to get fair coin flips from pairs of examples. In each case  $p$  denotes the probability that a fair flip is obtained from a pair. We show now how, given any polynomial-time, randomized learning algorithm  $A$ , a polynomial-time simulation  $B$  of  $A$  can be constructed that uses the described method for producing fair coin flips. We do this as follows. Let  $r$  denote an upper bound on the number of time steps to be used by algorithm  $A$  when the input parameters are  $\varepsilon$ ,  $\delta/2$ ,  $n$ , and  $s$ . Then  $r$  is clearly an upper bound on the number of unbiased coin flips needed by  $A$ .  $B$  will first try to get this many random bits for  $A$  and then run  $A$  using the bits and these input parameters. In case  $B$  fails to get enough bits ( $p$  is too small) then it will output the default hypothesis. A more detailed description of  $B$  follows.

Reset  $\varepsilon$  to  $\min(\varepsilon, \frac{1}{4})$ .  $B$  draws at least  $\max((8/\varepsilon) \ln(2/\delta), 2r/\varepsilon)$  pairs. If at least  $r$  coin flips are obtained then  $B$  uses these random bits to simulate  $A$  on the input parameters  $\varepsilon$ ,  $\delta/2$ , and  $s$  and outputs the hypothesis of  $A$ . Otherwise,  $B$  draws a sample of size at least  $24 \ln 2/\delta$  from the oracle that produced the pairs and outputs the default hypothesis corresponding the most frequent label in the sample (Case 1) or the most frequent point in the sample (Case 2).

It remains to be shown that with probability at least  $1 - \delta$  algorithm  $B$  outputs a hypothesis with error at most  $\varepsilon$ .

Case  $p \geq \varepsilon$ . The probability that fewer than  $r$  coin flips are obtained from  $m \geq \max((8/\varepsilon) \ln(2/\delta), 2r/\varepsilon)$  pairs is at most

$$LE(p, m, r) \leq LE(\varepsilon, m, r) \leq LE\left(\varepsilon, m, \frac{1}{2} m\varepsilon\right) \leq e^{-(1/8)m\varepsilon} \leq \frac{\delta}{2}.$$

Also the probability that  $A$ , when executed with the above input parameters, outputs a hypothesis with error larger than  $\varepsilon$  is at most  $\delta/2$ . Thus if  $p \geq \varepsilon$  then the total probability that  $B$  outputs a hypothesis with error at most  $\varepsilon$  is at least  $1 - \delta$ .

Before we address the remaining case observe the following fact. Let  $E$  be an event that has probability at least  $\frac{3}{4}$ . The probability that  $E$  occurs at most half the time in  $m \geq 24 \ln(2/\delta)$  independent draws is at most

$$LE\left(\frac{3}{4}, m, \frac{1}{2} m\right) \leq e^{-(1/24)m} \leq \frac{\delta}{2}.$$

Case  $p < \varepsilon$ . In this case the most likely label (respectively point) has probability  $q > 1 - \varepsilon$ . Thus the default hypothesis corresponding to the most frequent label (respectively point) has error (respectively positive error) less than  $\varepsilon$ . There are two possibilities in which  $B$  might produce a hypothesis of error greater than  $\varepsilon$ :  $A$  might produce such a hypothesis or the most frequent label (respectively point) in a sample of size  $m \geq 24 \ln(2/\delta)$  might not be the most likely label (respectively point). The first one occurs with probability at most  $\delta/2$  and the probability of the second one is also bounded by  $\delta/2$  since  $q > 1 - \varepsilon \geq \frac{3}{4}$ . Thus in case  $p < \varepsilon$  the total probability that  $B$  outputs a hypothesis of error at most  $\varepsilon$  is at least  $1 - \delta$ . ■

The above proof crucially relies on the fact that the learner receives a *sequence* of examples, drawn independently at random from the same distribution. The sequence may be produced by iteratively calling an oracle, or in the case of the functional model, such a sequence constitutes the input to the learning algorithm. Since each permutation of the sequence is equally likely, a sequence of  $m$  distinct examples provides the learner with  $\log(m!)$  "free" random bits. Thus it is not surprising that probabilistic learning algorithms can be converted to nonprobabilistic learning algorithms by using the order in which the examples appear as a source of randomness.

Suppose we changed the definition of sample to be a *multiset* of examples rather than a sequence of examples. Then it is possible that learnability

with respect to the models **functional**( $p_1, det$ ) and **functional**( $p_1, rand$ ) (for  $p_1$  as in Theorem 3.1) would no longer be equivalent.

LEMMA 3.6.

$$\begin{aligned} & \mathbf{one-oracle} \left( \log \frac{1}{\delta}, det, s\_known, always\_halts \right) \cap \mathbf{regular}_1 \\ & \subseteq \mathbf{functional} \left( \log \frac{1}{\delta}, det \right) \cap \mathbf{regular}_1. \end{aligned}$$

*Proof.* For any pair  $(\mathbf{C}, \mathbf{H}) \in \mathbf{one-oracle}(\log(1/\delta), det, s\_known, always\_halts) \cap \mathbf{regular}_1$ , let  $A$  be any polynomial algorithm for learning  $\mathbf{C}$  by  $\mathbf{H}$  in that model. We will construct an algorithm  $B$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  polynomially in the **functional**( $\log(1/\delta), det$ ) model. We know that there exist bounds  $S_A(\varepsilon, \delta, n, s) \leq T_A(\varepsilon, \delta, n, s)$  polynomial in  $1/\varepsilon, \log(1/\delta), n$ , and  $s$  such that if  $A$  is given  $\varepsilon, \delta$ , and  $s$  as input parameters and if the target concept is in  $C_{n,s}$ , then algorithm  $A$  halts after at most  $T_A$  time steps and after at most  $S_A$  calls to the oracle and with probability at least  $1 - \delta$  produces a hypothesis with error at most  $\varepsilon$ . Let  $p(n, x)$  be a polynomial in  $n$  and  $x$  that is monotonically increasing in  $x$  for positive values of  $n$  and  $x$  such that  $p(n, x) \geq T_A(1/x, 2^{-x}, n, x) \geq S_A(1/x, 2^{-x}, n, x)$ .

Algorithm  $B$  will be constructed as follows: First the algorithm reads the input and determines  $n$  and the number of examples  $m$ . Then it resets the input to the beginning. Now it chooses a positive integer  $x$  such that  $p(n, x) \leq m$  and  $p(n, 2x) > m$ . If it cannot do this because  $p(n, 1) > m$  then it halts with a default hypothesis. This concludes the preprocessing step. The time needed up to this point is clearly polynomial in the length of the input.

Next, algorithm  $B$  simulates the action of algorithm  $A$  as if the values of  $\varepsilon, \delta$ , and  $s$  given to  $A$  were  $1/x, 2^{-x}$ , and  $x$ , respectively. Whenever  $A$  requests an example from the oracle,  $B$  gives  $A$  the next input example instead. If  $A$  requests more input examples than are available, then  $B$  aborts the simulation of  $A$  and halts with a default hypothesis. We claim that the simulation satisfies Property 1. Specifically,

(a) there exists some polynomial  $q$  such that for any  $\varepsilon > 0$  and  $\delta, 0 < \delta < 1$ , if the sample contains  $m \geq q(1/\varepsilon, \log_2(1/\delta), n, s)$  examples, then with probability at least  $1 - \delta$  the hypothesis that  $B$  outputs will have error less than  $\varepsilon$ .

(b)  $B$  runs in time polynomial in the length of its input.

To see (a), choose  $q$  such that  $q(1/\varepsilon, \log_2(1/\delta), n, s) = p(n, 2(1/\varepsilon + \log_2(1/\delta) + s))$ . Now consider the case where  $m \geq q(1/\varepsilon, \log_2(1/\delta), n, s)$ . In this case,  $B$  will be able to find an  $x$  such that  $p(n, x) \leq m$  and  $p(n, 2x) > m$

and this  $x$  will be at least  $1/\varepsilon + \log_2(1/\delta) + s$ . Thus we have  $x \geq 1/\varepsilon$ ,  $x \geq \log_2(1/\delta)$ , and  $x \geq s$ , i.e.,  $1/x \leq \varepsilon$ ,  $2^{-x} \leq \delta$ , and  $s \leq x$ . Thus  $B$  will simulate  $A$  using values of  $\varepsilon$  and  $\delta$  which are no larger than the values of  $\varepsilon$  and  $\delta$  that we used in choosing the sample size  $m$  for  $B$ . Because  $B$  is receiving examples from some concept in  $C_{n,s}$  it can pass these examples to  $A$  which is expecting examples from a concept in  $C_{n,x}$  (since  $s \leq x$ ,  $C_{n,s} \subseteq C_{n,x}$ ). Furthermore, enough examples will be available for  $B$  to complete simulating  $A$ , since

$$S_A(1/x, 2^{-x}, n, x) \leq p(n, x) \leq m,$$

and with probability at least  $1 - 2^{-x} \geq 1 - \delta$  the simulation will output a hypothesis with error no more than  $1/x \leq \varepsilon$ .

To see (b), note that whenever an  $x$  is successfully found, and  $B$  simulates  $A$ , then the number of simulated time steps of  $A$  will be bounded by

$$T_A(1/x, 2^{-x}, n, x) \leq p(x, n) \leq m.$$

Thus the total time required by algorithm  $B$  will be polynomial in the length of the input. ■

LEMMA 3.7.

$$\begin{aligned} & \text{functional}(\text{fixed } \delta, \text{rand}) \cap \text{regular}_1 \\ \subseteq & \text{one-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1. \end{aligned}$$

*Proof.* Let  $(\mathbf{C}, \mathbf{H})$  be any pair in  $\text{functional}(\text{fixed } \delta, \text{rand}) \cap \text{regular}_1$ . Consider an algorithm  $A$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  in that model for  $\delta$  fixed at  $\delta_0$ . We will construct an algorithm  $B$  for learning  $\mathbf{C}$  by  $\mathbf{H}$  in the **one-oracle**(fixed  $\delta$ , rand,  $s$ -known, always-halts) model. Let  $S_A(\varepsilon, n, s)$  be the sample complexity of  $A$ . Let  $p$  be any polynomial in three variables such that for all positive  $\varepsilon, n$ , and  $s$  we have  $p(1/\varepsilon, n, s) \geq S_A(\varepsilon, n, s)$ . Algorithm  $B$  will receive  $\varepsilon$  and  $s$  as input. Algorithm  $B$  first requests  $\lfloor p(1/\varepsilon, n, s) \rfloor$  examples from the oracle. It then simulates algorithm  $A$  with these examples as input. With this many examples algorithm  $A$  will with probability at least  $1 - \delta_0$  find a hypothesis with error at most  $\varepsilon$ , as desired. Since algorithm  $A$  runs in time polynomial in the length of its input, algorithm  $B$  runs in time polynomial in  $1/\varepsilon, n$ , and  $s$ . ■

LEMMA 3.8.

$$\begin{aligned} & \text{one-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{usually-halts}) \cap \text{regular}_1 \\ \subseteq & \text{one-oracle}(\text{fixed } \delta, \text{rand}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1 \end{aligned}$$

and

$$\begin{aligned} & \text{two-oracle}(\text{fixed } \delta, \text{rand}, s\_known, \text{usually\_halts}) \cap \text{regular}_1 \\ & \subseteq \text{two-oracle}(\text{fixed } \delta, \text{rand}, s\_known, \text{always\_halts}) \cap \text{regular}_1. \end{aligned}$$

*Proof.* We state the proof for the **one-oracle** model. The proof for the **two-oracle** model is essentially identical. Let  $(\mathbf{C}, \mathbf{H})$  be any pair in  $\text{one-oracle}(\text{fixed } \delta, \text{rand}, s\_known, \text{usually\_halts}) \cap \text{regular}_1$ . Consider an algorithm  $A$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  in that model for some fixed  $\delta = \delta_0$ . We construct an algorithm  $B$  for learning  $\mathbf{C}$  by  $\mathbf{H}$  in the  $\text{one-oracle}(\text{fixed } \delta, \text{rand}, s\_known, \text{always\_halts})$  model. Let  $p$  be a polynomial bound for algorithm  $A$  as in Property 2'. Algorithm  $B$  receives  $\varepsilon$  and  $s$  as input. Algorithm  $B$  simulates algorithm  $A$ , keeping track of the total number of time steps of  $A$  that have been simulated. If this number ever exceeds  $\lfloor p(1/\varepsilon, n, s) \rfloor$ , then algorithm  $B$  terminates and outputs a default hypothesis. With probability at least  $1 - \delta_0$ , the simulation of algorithm  $A$  halts with a hypothesis with error at most  $\varepsilon$  after no more than  $p(1/\varepsilon, n, s)$  time steps of  $A$  have been simulated. In this case  $B$  halts with a hypothesis with error no more than  $\varepsilon$ . In every case, the time used by  $B$  is polynomial in  $1/\varepsilon$ ,  $n$ , and  $s$ , as desired. ■

One final equivalence (Lemma 3.10) is needed to complete the proof of Theorem 3.1. To show this equivalence we need the following algorithm for testing hypotheses.

**ALGORITHM TEST1**( $\varepsilon, \delta, i, h$ ). This is a hypothesis testing algorithm with parameters  $\varepsilon, \delta, i$ , and  $h$  that runs in the **one-oracle** model. It makes  $\lceil (32/\varepsilon)(i \ln 2 + \ln 2/\delta) \rceil$  calls to the oracle to test hypothesis  $h$ . It accepts the hypothesis if the hypothesis is wrong on no more than a fraction  $\frac{3}{4}\varepsilon$  of the examples returned by the oracle, and rejects it otherwise.

**LEMMA 3.9.** *The test TEST1*( $\varepsilon, \delta, i, h$ ) *has the property that*

- (1) *When  $h$  has error greater than or equal to  $\varepsilon$ , the probability is at most  $\delta/2^{i+1}$  that the test will accept  $h$ .*
- (2) *When  $h$  has error at most  $\varepsilon/2$ , the probability is at most  $\delta/2^{i+1}$  that the algorithm will reject  $h$ .*

*Proof.* If the hypothesis has error  $p$ , then errors of the kind mentioned in (1) above occur only if  $p \geq \varepsilon$ , and in that case the probability of such an error is bounded by  $LE(p, m, \frac{3}{4}\varepsilon m) = LE(p, m, (1 - \alpha)mp)$ , for some  $\alpha \geq \frac{1}{4}$ . This is bounded by  $e^{-m\alpha\varepsilon/32}$ . Plugging in the value for  $m$  we get a bound of  $\delta/2^{i+1}$ .

Errors of the kind mentioned in (2) occur only if  $p \leq \varepsilon/2$  and in that case the probability of such an error is bounded by  $GE(p, m, \frac{3}{4}\varepsilon m) \leq GE(\varepsilon/2, m, (1 + 1/2)m(\varepsilon/2)) \leq e^{-m\varepsilon/24}$ . Again this gives the desired bound. ■

A two-oracle version  $TEST2(\varepsilon, \delta, i, h)$  can be constructed by running the one-oracle version  $TEST1(\varepsilon, \delta/2, i, h)$  twice, once using the positive oracle and once using the negative oracle.  $TEST2$  accepts  $h$  if and only if both of the above calls to  $TEST1$  accept  $h$ . The above lemma holds for  $TEST2$  as well.

LEMMA 3.10 (see also Benedek and Itai, 1988b, and Linial *et al.*, 1988).

$$\begin{aligned} & \text{one-oracle} \left( \log \frac{1}{\delta}, \text{det}, s\text{-known}, \text{always-halts} \right) \cap \text{regular}_1 \\ & \subseteq \text{one-oracle} \left( \log \frac{1}{\delta}, \text{det}, s\text{-unknown}, \text{usually-halts} \right) \cap \text{regular}_1 \end{aligned}$$

and

$$\begin{aligned} & \text{two-oracle} \left( \log \frac{1}{\delta}, \text{det}, s\text{-known}, \text{always-halts} \right) \cap \text{regular}_1 \\ & \subseteq \text{two-oracle} \left( \log \frac{1}{\delta}, \text{det}, s\text{-unknown}, \text{usually-halts} \right) \cap \text{regular}_1. \end{aligned}$$

*Proof.* We state the proof for the **one-oracle** model. The proof of the **two-oracle** model is identical except for replacing  $TEST1$  by  $TEST2$ . Let  $(\mathbf{C}, \mathbf{H})$  be any pair in  $\text{one-oracle}(\log 1/\delta, \text{det}, s\text{-known}, \text{always-halts}) \cap \text{regular}_1$ . Consider an algorithm  $A$  that learns  $\mathbf{C}$  by  $\mathbf{H}$  in that model. We construct an algorithm  $B$  for learning  $\mathbf{C}$  by  $\mathbf{H}$  in the  $\text{one-oracle}(\log 1/\delta, \text{det}, s\text{-unknown}, \text{usually-halts})$  model. Let  $p$  be a polynomial such that for all positive  $\varepsilon, n$ , and  $s$  we have  $p(1/\varepsilon, n, s) \geq S_A(\varepsilon, \frac{1}{2}, n, s)$  and  $p(1/\varepsilon, n, s) \geq T_A(\varepsilon, \frac{1}{2}, n, s)$ . We construct an oracle algorithm  $B$  that receives  $\varepsilon$  and  $\delta$  as input. Since  $s$  is not known to the algorithm, the algorithm performs the following procedure repeatedly, gradually increasing its guess of the size of  $s$ . Specifically, at the  $i$ th repetition (calling the first time the procedure is performed the first repetition), algorithm  $B$  lets  $\tilde{s} = \lfloor 2^{(i-1)/\ln(2/\delta)} \rfloor$ . For each repetition,  $B$  simulates the action of  $A$  as if the size parameter given to  $A$  were  $\tilde{s}$ , using the value  $\frac{1}{2}$  for  $\delta$  and using for  $\varepsilon$  one-half of the value of  $\varepsilon$  given to  $B$ . When this simulation finishes, producing a hypothesis  $h$ , algorithm  $B$  tests the hypothesis. For the  $i$ th repetition,  $B$  uses the testing procedure  $TEST1(\varepsilon, \delta, i, h)$  to decide whether or not to accept  $h$ . If it accepts  $h$  then it halts with  $h$  as its hypothesis.

If it rejects  $h$  then it continues on to the next repetition. The procedure *TEST1* has been constructed so that no matter how many repetitions are performed the total probability that a hypothesis with error greater than  $\varepsilon$  is accepted is bounded by  $\delta/2$ .

We next show that with high probability the algorithm halts in polynomial time. Let  $j = \lceil \ln(2/\delta)/\ln(8/5) \rceil$  and let

$$j' = \lceil (\ln(2/\delta)) \log_2 s \rceil + j \leq (\ln(2/\delta)) \log_2 s + \frac{\ln(2/\delta)}{\ln(8/5)} + 2.$$

We claim that with probability at least  $1 - \delta/2$  the algorithm will halt after at most  $j'$  repetitions. To see this note that for all  $i \geq \lceil 1 + (\ln(2/\delta)) \log_2 s \rceil$ , the algorithm will choose  $\tilde{s} \geq s$  at the  $i$ th repetition. After any repetition for which  $\tilde{s} \geq s$ , the probability will be at least one-half that the resulting hypothesis will have error no more than  $\varepsilon/2$ . A hypothesis with error at most  $\varepsilon/2$  will be accepted by *TEST1* with probability at least  $\frac{3}{4}$  (by the previous lemma). Therefore after such a repetition the probability is at least  $\frac{3}{8}$  that the algorithm will halt. For the algorithm to fail to halt after a total of  $j'$  repetitions, it must fail to halt after each of  $j$  repetitions for which  $\tilde{s} \geq s$ , which will occur with probability at most  $(\frac{5}{8})^j \leq \delta/2$ . This proves the claim.

We now show that if the algorithm halts after no more than  $j'$  repetitions then it runs in polynomial time. The number of time steps of  $A$  simulated at the  $i$ th repetition is bounded by  $\lceil p(2/\varepsilon, n, \lfloor 2^{(i-1)/\ln(2/\delta)} \rfloor) \rceil \leq 2^{k(i-1)/\ln(2/\delta)} p'(1/\varepsilon, n)$  for some constant  $k$  and polynomial  $p'$ . Thus the total number of simulated time steps for  $j'$  repetitions will be bounded by

$$\begin{aligned} p' \left( \frac{1}{\varepsilon}, n \right) \sum_{i=0}^{j'-1} 2^{ki/\ln(2/\delta)} &= p' \left( \frac{1}{\varepsilon}, n \right) \frac{2^{kj'/\ln(2/\delta)} - 1}{2^{k/\ln(2/\delta)} - 1} \\ &\leq p' \left( \frac{1}{\varepsilon}, n \right) \frac{2^{k \log_2 s + k/\ln(8/5) + 2k/\ln(2/\delta)} - 1}{2^{k/\ln(2/\delta)} - 1} \\ &\leq p' \left( \frac{1}{\varepsilon}, n \right) \frac{\ln(2/\delta) (2^{k \log_2 s + k/\ln(8/5) + 2k/\ln(2/\delta)} - 1)}{k \ln 2} \end{aligned}$$

which is polynomial in  $1/\varepsilon$ ,  $\log(1/\delta)$ ,  $n$ , and  $s$ . Thus the time used by  $B$  to simulate  $A$  will be polynomial in these variables.

The time for the hypothesis testing grows in proportion to the number of examples used in the tests. For  $j'$  repetitions, this is bounded by  $\lceil (32/\varepsilon)(j' \ln 2 + \ln(2/\delta)) \rceil j'$ . This is polynomial in  $1/\varepsilon$ ,  $\log(1/\delta)$ ,  $n$ , and  $s$ . Thus the total time needed by algorithm  $B$  for  $j'$  repetitions is polynomial in these variables.

We have shown that that probability is at most  $\delta/2$  that the algorithm

