Machine Learning, 5, 197–227 (1990)

The Strength of Weak Learnability

ROBERT E. SCHAPIRE (rs@theory.lcs.mit.edu)
MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139

Abstract. This paper addresses the problem of improving the accuracy of an hypothesis output by a learning algorithm in the distribution-free (*PAC*) learning model. A concept class is *learnable* (or *strongly learnable*) if, given access to a source of examples of the unknown concept, the learner with high probability is able to output an hypothesis that is correct on all but an arbitrarily small fraction of the instances. The concept class is *weakly learnable* if the learner can produce an hypothesis that performs only slightly better than random guessing. In this paper, it is shown that these two notions of learnability are equivalent.

A method is described for converting a weak learning algorithm into one that achieves arbitrarily high accuracy. This construction may have practical applications as a tool for efficiently converting a mediocre learning algorithm into one that performs extremely well. In addition, the construction has some interesting theoretical consequences, including a set of general upper bounds on the complexity of any strong learning algorithm as a function of the allowed error ϵ.

Keywords. Machine learning, learning from examples, learnability theory, PAC learning, polynomial-time identification.

1. Introduction

Since Valiant's pioneering paper (1984), interest has flourished in the so-called *distribution-free* or *probably approximately correct* (PAC) model of learning. In this model, the learner tries to identify an unknown concept based on randomly chosen examples of the concept. Examples are chosen according to a fixed but unknown and arbitrary distribution on the space of instances. The learner's task is to find an hypothesis or prediction rule of his own that correctly classifies new instances as positive or negative examples of the concept. With high probability, the hypothesis must be correct for all but an arbitrarily small fraction of the instances.

Often, the inference task includes a requirement that the output hypothesis be of a specified form. In this paper, however, we will instead be concerned with a representation-independent model of learning in which the learner may output any hypothesis that can be used to classify instances in polynomial time.

A class of concepts is *learnable* (or *strongly learnable*) if there exists a polynomial-time algorithm that achieves low error with high confidence for all concepts in the class. A weaker model of learnability, called *weak learnability*, drops the requirement that the learner be able to achieve arbitrarily high accuracy; a weak learning algorithm need only output an hypothesis that performs slightly better (by an inverse polynomial) than random guessing. The notion of weak learnability was introduced by Kearns and Valiant (1988; 1989) who left open the question of whether the notions of strong and weak learnability are equivalent. This question was termed the *hypothesis boosting problem* since showing the notions are equivalent requires a method for boosting the low accuracy of a weak learning algorithm's hypotheses.

Kearns (1988), considering the hypothesis boosting problem, gives a convincing argument discrediting the natural approach of trying to boost the accuracy of a weak learning algorithm by running the procedure many times and taking the "majority vote" of the output hypotheses. Also, Kearns and Valiant (1989) show that, under a uniform distribution on the instance space, monotone Boolean functions are weakly, but not strongly, learnable. This shows that strong and weak learnability are *not* equivalent when certain restrictions are placed on the instance space distribution. Thus, it did not seem implausible that the strong and weak learning models would prove to be inequivalent for unrestricted distributions as well.

Nevertheless, in this paper, the hypothesis boosting question is answered in the affirmative. The main result is a proof of the perhaps surprising equivalence of strong and weak learnability.

This result may have significant applications as a tool for proving that a concept class is learnable since, in the future, it will suffice to find an algorithm correct on only, say, 51% of the instances (for all distributions). Alternatively, in its negative contrapositive form, the result says that if a concept class cannot be learned with accuracy 99.9%, then we cannot hope to do even slightly better than guessing on the class (for some distribution).

The proof presented here is constructive; an explicit method is described for directly converting a weak learning algorithm into one that achieves arbitrary accuracy. The construction uses *filtering* to modify the distribution of examples in such a way as to force the weak learning algorithm to focus on the harder-to-learn parts of the distribution. Thus, the distribution-free nature of the learning model is fully exploited.

An immediate corollary of the main result is the equivalence of strong and *group* learnability. A group-learning algorithm need only output an hypothesis capable of classifying large groups of instances, all of which are either positive or negative. The notion of group learnability was considered by Kearns, Li, Pitt and Valiant (1987), and was shown to be equivalent to weak learnability by Kearns and Valiant (1989). The result also extends those of Haussler, Kearns, Littlestone and Warmuth (1988) which prove the equivalence of numerous relaxations and variations on the basic PAC-learning model; both weak and group learnability are added to this class of equivalent learning models. The relevance of the main result to a number of other learning models is also considered in this paper.

An interesting and unexpected consequence of the construction is a proof that any strong learning algorithm outputting hypotheses whose length (and thus whose time to evaluate) depends on the allowed error ϵ can be modified to output hypotheses of length only polynomial in $\log(1/\epsilon)$. Thus, any learning algorithm can be converted into one whose ouptut hypotheses do not become significantly more complex as the error tolerance is lowered.

Put in other terms, this bound implies that a sequence of labeled examples of a learnable concept can, in a sense, be efficiently *compressed* into a far more compact form—that is, into a rule or hypothesis consistent with the labels of the examples. In particular, it is shown that a sample of size m can be compressed into a rule of size only poly-logarithmic in m. In fact, in the discrete case, the size of the output hypothesis is entirely independent of m. This provides a partial converse to *Occam's Razor*, a result of Blumer, Ehrenfeucht, Haussler and Warmuth (1987) stating that the existence of such a compression algorithm implies the learnability of the concept class. This also complements the results of Board and Pitt (1990) who also provide a partial converse to Occam's Razor, but of a somewhat different flavor. Finally, this result yields a strong bound on the sample size needed to learn a discrete concept class.

This bound on the size of the output hypothesis also implies the hardness of learning any concept class not evaluatable by a family of small circuits. For example, this shows that pattern languages—a class of languages considered previously by Angluin (1980) and others—are unlearnable assuming only that NP/poly $\neq$ P/poly. This is the first representation-independent hardness result not based on cryptographic assumptions. The bound also shows that, for any function not computable by polynomial-size circuits, there exists a distribution on the function's domain over which the function cannot be even roughly approximated by a family of small circuits.

In addition to the bound on hypothesis size, the construction implies a set of general bounds on the dependence on ϵ of the time, sample and space complexity needed to efficiently learn any learnable concept class. Most surprising is a proof that there exists for every learnable concept class an efficient algorithm requiring space only poly-logarithmic in $1/\epsilon$. Because the size of the sample needed to learn with this accuracy is in general $\Omega(1/\epsilon)$, this means, for example, that far less space is required to learn than would be necessary to store the entire sample. Since most of the known learning algorithms work in exactly this manner—that is, by storing a large sample and finding an hypothesis consistent with it—this implies a dramatic savings of memory for a whole class of algorithms (though possibly at the cost of requiring a larger sample).

Such general complexity bounds have implications for the *on-line* learning model as well. In this model, the learner is presented one instance at a time in a series of trials. As each is received, the learner tries to predict the true classification of the new instance, attempting to minimize the number of mistakes, or prediction errors.

Translating the bounds described above into the on-line model, it is shown that, for every learnable concept class, there exists an on-line algorithm whose space requirements are quite modest in comparison to the number of examples seen so far. In particular, the space needed on the first m trials is only poly-logarithmic in m. Such space efficient on-line algorithms are of particular interest because they capture the notion of an incremental algorithm forced by its limited memory to explicitly generalize or abstract from the data observed. Also, these results on the space-efficiency of batch and on-line algorithms extend the work of others interested in this problem, including Boucheron and Sallantin (1988), Floyd (1989), and Haussler (1988). In particular, these results solve an open problem proposed by Haussler, Littlestone and Warmuth (1987).

An interesting bound is also derived on the expected number of mistakes made on the first m trials. It is shown that, if a concept class is learnable, then there exists an on-line algorithm for the class for which this expectation is bounded by a polynomial in $\log m$. Thus, for large m, we expect an extremely small fraction of the first m predictions to be incorrect. This result answers another open question given by Haussler, Littlestone and Warmuth (1987), and significantly improves a similar bound given in their paper (as well as their paper with Kearns (1988)) of m^{α} for some constant $\alpha < 1$.

2. Preliminaries

We begin with a description of the distribution-free learning model. A *concept* c is a Boolean function on some domain of *instances*. A *concept class* $\boldsymbol{C}$ is a family of concepts. Often, $\boldsymbol{C}$ is decomposed into subclasses $\boldsymbol{C}_n$ indexed by a parameter n. That is, $\boldsymbol{C} = \cup_{n \geq 1} \boldsymbol{C}_n$,

and all the concepts in $\boldsymbol{C}_n$ have a common domain X_n. We assume each instance in X_n has encoded length bounded by a polynomial in n, and we let $X = \cup_{n \geq 1} X_n$. Also, we associate with each concept c its *size* s, typically a measure of the length of c's representation under some encoding scheme on the concepts in $\boldsymbol{C}$.

For example, the concept class $\boldsymbol{C}$ might consist of all functions computed by Boolean formulas. In this case, $\boldsymbol{C}_n$ is the set of all functions computed by a Boolean formula on n variables, X_n is the set $\{0, 1\}^n$ of all assignments to the n variables, and the size of a concept c in $\boldsymbol{C}$ is the length of the shortest Boolean formula that computes the function c.

The learner is assumed to have access to a source EX of examples. Each time oracle EX is called, one instance is randomly and independently chosen from X_n according to some fixed but unknown and arbitrary distribution D. The oracle returns the chosen instance v, along with a label indicating the value $c(v)$ of the instance under the unknown *target concept* $c \in \boldsymbol{C}_n$. Such a labeled instance is called an *example*. We assume EX runs in unit time.

Given access to EX, the learning algorithm runs for a time and finally outputs an *hypothesis* h, a prediction rule on X_n. In this paper, we make no restrictions on h other than that there exist a (possibly probabilistic) polynomial time algorithm that, given h and an instance v, computes $h(v)$, h's prediction on v.

We write $\Pr_{v \in D}[\pi(v)]$ to indicate the probability of predicate π holding on instances v drawn from X_n according to distribution D. To accommodate probabilistic hypotheses, we will find it useful to regard $\pi(v)$ as a Bernoulli random variable. For example, $\Pr[h(v) \neq c(v)]$ is the chance that hypothesis h (which may be randomized) will misclassify some particular instance v. In contrast, the quantity $\Pr_{v \in D}[h(v) \neq c(v)]$ is the probability that h will misclassify an instance v chosen at random according to distribution D. Note that this last probability is taken over both the random choice of v, and any random bits used by h.

In general, assuming independence, we have

$$\Pr_{v \in D}[\pi(v)] = \sum_{v \in X_n} D(v) \Pr[\pi(v)]$$

where $D(v)$ is the probability of instance v being chosen under D. (Technically, this formula is valid only when X_n is discrete. To handle general domains, the summation would need to be replaced by an appropriate integral, and D by a probability measure on the domain. To simplify the presentation, we will assume that X_n is discrete, and omit the extension of these results to general domains; this extension simply mimics the discrete case.)

The probability $\Pr_{v \in D}[h(v) \neq c(v)]$ is called the *error* of h on c under D; if the error is no more than ϵ, then we say h is *ϵ-close* to the target concept c under D. The quantity $\Pr_{v \in D}[h(v) = c(v)]$ is the *accuracy* of h on c under D.

We say that a concept class $\boldsymbol{C}$ is *learnable*, or *strongly learnable*, if there exists an algorithm A such that for all $n \geq 1$, for all target concepts $c \in \boldsymbol{C}_n$, for all distributions D on X_n, and for all $0 < \epsilon, \delta \leq 1$, algorithm A, given parameters n, ϵ, δ, the size s of c, and access to oracle EX, runs in time polynomial in n, s, $1/\epsilon$ and $1/\delta$, and outputs an hypothesis h that with probability at least $1 - \delta$ is ϵ-close to c under D. There are many other equivalent notions of learnability, including polynomial predictability (Haussler, Kearns, Littlestone and Warmuth, 1988). Also, note that other authors have sometimes used the term *learnable* to mean something slightly different.

Kearns and Valiant (1989) introduced a weaker form of learnability in which the error ϵ cannot necessarily be made arbitrarily small. A concept class $\boldsymbol{C}$ is *weakly learnable* if there exists a polynomial p and an algorithm A such that for all $n \geq 1$, for all target concepts $c \in \boldsymbol{C}_n$, for all distributions D on X_n, and for all $0 < \delta \leq 1$, algorithm A, given parameters n, δ, the size s of c, and access to oracle EX, runs in time polynomial in n, s and $1/\delta$, and outputs an hypothesis h that with probability at least $1 - \delta$ is $(1/2 - 1/p(n, s))$-close to c under D. In other words, a weak learning algorithm produces a prediction rule that performs just slightly better than random guessing.

3. The equivalence of strong and weak learnability

The main result of this paper is a proof that strong and weak learnability are equivalent notions.

THEOREM 1. A concept class $\boldsymbol{C}$ is weakly learnable if and only if it is strongly learnable.

That strong learnability implies weak learnability is trivial. The remainder of this section is devoted to a proof of the converse. We assume then that some concept class $\boldsymbol{C}$ is weakly learnable and show how to build a strong learning algorithm around a weak one.

We begin with a description of a technique by which the accuracy of any algorithm can be boosted by a small but significant amount. Later, we will show how this mechanism can be applied recursively to make the error arbitrarily small.

3.1. The hypothesis boosting mechanism

Let A be an algorithm that produces with high-probability an hypothesis α-close to the target concept c. We sketch an algorithm A' that simulates A on three different distributions, and outputs an hypothesis significantly closer to c.

Let EX be the given examples oracle, and let D be the distribution on X_n induced by EX. The algorithm A' begins by simulating A on the original distribution $D_1 = D$, using the given oracle $EX_1 = EX$. Let h_1 be the hypothesis output by A.

Intuitively, A has found some weak advantage on the original distribution; this advantage is expressed by h_1. To force A to learn more about the "harder" parts of the distribution, we must somehow destroy this advantage. To do so, A' creates a new distribution D_2 under which an instance chosen according to D_2 has a roughly equal chance of being correctly or incorrectly classified by h_1. The distribution D_2 is simulated by filtering the examples chosen according to D by EX. To simulate D_2, a new examples oracle EX_2 is constructed. When asked for an instance, EX_2 first flips a fair coin: if the result is *heads*, then EX_2 requests examples from EX until one is chosen for which $h_1(v) = c(v)$; otherwise, EX_2 waits for an instance to be chosen for which $h_1(v) \neq c(v)$. (Later we show how to prevent EX_2 from having to wait too long in either of these loops for a desired instance). The algorithm A is again simulated, this time providing A with examples chosen by EX_2 according to D_2. Let h_2 be the output hypothesis.

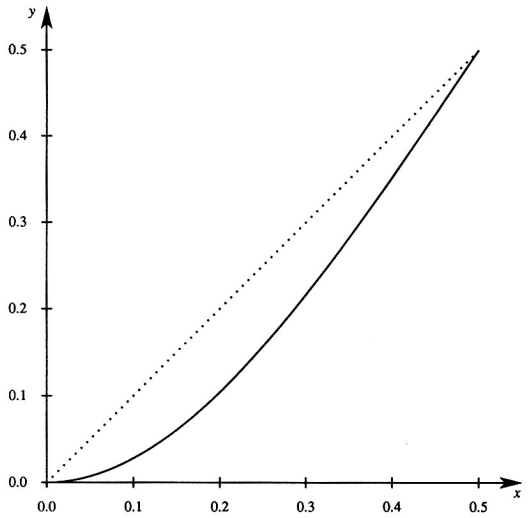

Figure 1. A graph of the function $g(x) = 3x^2 - 2x^3$.

Finally, D_3 is constructed by filtering from D those instances on which h_1 and h_2 agree. That is, a third oracle EX_3 simulates the choice of an instance according to D_3 by requesting instances from EX until one is found for which $h_1(v) \neq h_2(v)$. (Again, we will later show how to limit the time spent waiting in this loop for a desired instance.) For a third time, algorithm A is simulated with examples drawn this time by EX_3, producing hypothesis h_3.

At last, A' outputs its hypothesis h: given an instance v, if $h_1(v) = h_2(v)$ then h predicts the agreed upon value; otherwise, h predicts $h_3(v)$. (In other words, h takes the "majority vote" of h_1, h_2 and h_3.) Later, we show that h's error is bounded by $g(\alpha) \equiv 3\alpha^2 - 2\alpha^3$. This quantity is significantly smaller than the original error α, as can be seen from its graph depicted in Figure 1. (The solid curve is the function g, and, for comparison, the dotted line shows a graph of the identity function.)

3.2. A strong learning algorithm

An idea that follows naturally is to treat the previously described procedure as a subroutine for recursively boosting the accuracy of weaker hypotheses. The procedure is given a desired error bound ϵ and a confidence parameter δ, and constructs an ϵ-close hypothesis from weaker, recursively computed hypotheses. If $\epsilon \geq 1/2 - 1/p(n, s)$ then an assumed weak learning algorithm can be used to find the desired hypothesis; otherwise, an ϵ-close hypothesis is computed recursively by calling the subroutine with ϵ set to $g^{-1}(\epsilon)$.

Unfortunately, this scheme by itself does not quite work due to a technical difficulty: because of the way EX_2 and EX_3 are constructed, examples may be required from a very small portion of the original distribution. If this happens, the time spent waiting for an example to be chosen from this region may be great. Nevertheless, we will see that this

```
Learn(ε, δ, EX)

Input:  error parameter ε
        confidence parameter δ
        examples oracle EX
        (implicit) size parameters s and n

Return: hypothesis h that is ε-close to the target concept c with probability ≥ 1 − δ

Procedure:
  if ε ≥ 1/2 − 1/p(n, s) then return WeakLearn(δ, EX)
  α ← g^{-1}(ε)

  EX_1 ← EX
  h_1 ← Learn(α, δ/5, EX_1)
  τ_1 ← ε/3
  let â_1 be an estimate of a_1 = Pr_{v∈D}[h_1(v) ≠ c(v)]:
     choose a sample sufficiently large that |a_1 − â_1| ≤ τ_1 with probability ≥ 1 − δ/5
  if â_1 ≤ ε − τ_1 then return h_1

  defun EX_2()
     { flip coin
       if heads, return the first instance v from EX for which h_1(v) = c(v)
       else      return the first instance v from EX for which h_1(v) ≠ c(v) }
  h_2 ← Learn(α, δ/5, EX_2)
  τ_2 ← (1 − 2α)ε/8
  let ê be an estimate of e = Pr_{v∈D}[h_2(v) ≠ c(v)]:
     choose a sample sufficiently large that |e − ê| ≤ τ_2 with probability ≥ 1 − δ/5
  if ê ≤ ε − τ_2 then return h_2

  defun EX_3()
     { return the first instance v from EX for which h_1(v) ≠ h_2(v) }
  h_3 ← Learn(α, δ/5, EX_3)

  defun h(v)
     { b_1 ← h_1(v), b_2 ← h_2(v)
       if b_1 = b_2 then return b_1
       else              return h_3(v) }
  return h
```

Figure 2. A strong learning algorithm Learn.

difficulty can be overcome by explicitly checking that the errors of hypotheses h_1 and h_2 on D are not too small.

Figure 2 shows a detailed sketch of the resulting strong learning algorithm Learn. The procedure takes an error parameter ϵ and a confidence parameter δ, and is also provided with an examples oracle *EX*. The procedure is required to return an hypothesis whose error is at most ϵ with probability at least $1 - \delta$. In the figure, p is a polynomial and WeakLearn(δ, *EX*) is an assumed weak learning procedure that outputs an hypothesis $(1/2 - 1/p(n, s))$-close to the target concept c with probability at least $1 - \delta$. As above, $g(\alpha)$

is the function $3\alpha^2 - 2\alpha^3$, and the variable α is set to the value $g^{-1}(\epsilon)$. Also, the quantities $\hat{a}_1$ and $\hat{e}$ are estimates of the errors of h_1 and h_2 under the given distribution D. These estimates are made with error tolerances τ_1 and τ_2 (defined in the figure), and are computed in the obvious manner based on samples drawn from EX; the required size of these samples can be determined, for instance, using Chernoff bounds. The parameters s and n are assumed to be known globally.

Note that Learn is a procedure taking as one of its inputs a function (EX) and returning as output another function (h, a hypothesis, which is treated like a procedure). Furthermore, to simulate new example oracles, Learn must have a means of dynamically defining new procedures (as is allowed, for instance, by most Lisp-like languages). Therefore, in the figure, we have used the somewhat nonstandard keyword **defun** to denote the definition of a new function; its syntax calls for a name for the procedure, followed by a parenthesized list of arguments, and the body indented in braces. Static scoping is assumed.

Learn works by recursively boosting the accuracy of its hypotheses. Learn typically calls itself three times using the three simulated example oracles described in the preceding section. On each recursive call, the required error bound of the constructed hypotheses comes closer to 1/2; when this bound reaches $1/2 - 1/p(n, s)$, the weak learning algorithm WeakLearn can be used.

The procedure takes measures to limit the run time of the simulated oracles it provides on recursive calls. When Learn calls itself a second time to find h_2, the expected number of iterations of EX_2 to find an example depends on the error of h_1, which is estimated by $\hat{a}_1$. If h_1 already has the desired accuracy $1 - \epsilon$, then there is no need to find h_2 and h_3 since h_1 is a sufficiently good hypothesis; otherwise, if $a_1 = \Omega(\epsilon)$, then it can be shown that EX_2 will not loop too long to find an instance. Similarly, when Learn calls itself to find h_3, the expected number of iterations of EX_3 depends on how often h_1 and h_2 disagree, which we will see is in turn a function of the error of h_2 on the original distribution D. If this error e (which is estimated by $\hat{e}$) is small, then h_2 is a good hypothesis and is returned by Learn. Otherwise, it will be shown that EX_3 also will not run for too long.

3.3. Correctness

We show in this section that the algorithm is correct in the following sense:

THEOREM 2. For $0 < \epsilon < 1/2$ and for $0 < \delta \leq 1$, the hypothesis returned by calling Learn(ϵ, δ, EX) is ϵ-close to the target concept with probability at least $1 - \delta$.

Proof. In proving this theorem, we will find it useful to assume that nothing "goes wrong" throughout the execution of Learn. More specifically, we will say that Learn has a *good run* if every hypothesis returned by WeakLearn is indeed $(1/2 - 1/p(n, s))$-close to the target concept, and if every statistical estimate (i.e., of the quantities a_1 and e) is obtained with the required accuracy. We will then argue inductively on the depth of the recursion that if Learn has a good run then the output hypothesis is ϵ-close to the target concept, and furthermore, that the probability of a good run is at least $1 - \delta$. Together, these facts clearly imply the theorem's statement.

The base case that $\epsilon \geq 1/2 - 1/p(n, s)$ is trivially handled using our assumptions about WeakLearn.

In the general case, by inductive hypothesis, each of the three (or fewer) recursive calls to Learn are good runs with probability at least $1 - \delta/5$. Moreover, each of the estimates $\hat{a}_1$ and $\hat{e}$ has the desired accuracy with probability at least $1 - \delta/5$. Thus, the chance of a good run is at least the chance that all five of these events occur, which is at least $1 - \delta$.

It remains then only to show that on a good run the output hypothesis has error at most ϵ.

An easy special case is that $\hat{a}_1$ or $\hat{e}$ is found to be smaller than $\epsilon - \tau_1$ or $\epsilon - \tau_2$, respectively. In either case, it follows immediately, due to the accuracy with which a_1 and e are assumed to have been estimated, that the returned hypothesis is ϵ-close to the target concept.

Otherwise, in the general case, all three sub-hypotheses must be found and combined. Let a_i be the error of h_i under D_i. Here, D is the distribution of the provided oracle EX, and D_i is the distribution induced by oracle EX_i on the ith recursive call ($i = 1, 2, 3$). By inductive hypothesis, each $a_i \leq \alpha$.

In the special case that all hypotheses are deterministic, the distributions D_1 and D_2 can be depicted schematically as shown in Figure 3. The figure shows the portion of each distribution on which the hypotheses h_1 and h_2 agree with the target concept c. For each distribution, the top crosshatched bar represents the relative fraction of the instance space on which h_1 agrees with c; the bottom striped bar represents those instances on which h_2 agrees with c. Although only valid for deterministic hypotheses, this figure may be helpful for motivating one's intuition in what follows.

Let $p_i(v) = \Pr[h_i(v) \neq c(v)]$ be the chance that some fixed instance v is misclassified by h_i. (Recall that hypotheses may be randomized, and therefore it is necessary to consider the *probability* that a particular fixed instance is misclassified.) Similarly, let $q(v) = \Pr[h_1(v) \neq h_2(v)]$ be the chance that v is classified differently by h_1 and h_2. Also define w, x, y, and z as follows:

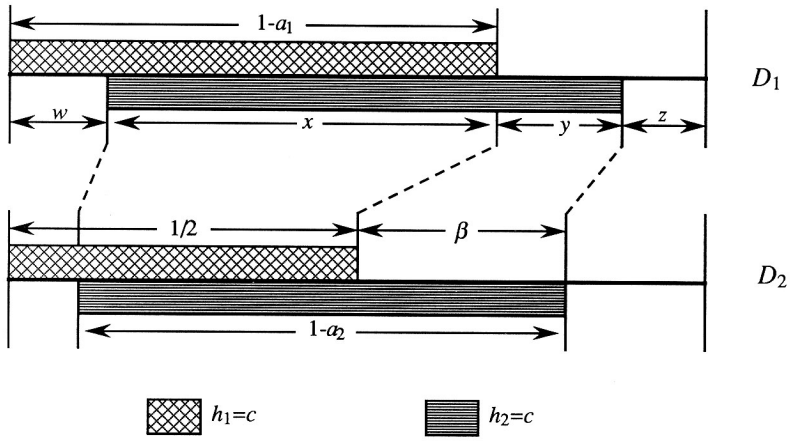

Figure 3. The distributions D_1 and D_2.

$$w = \Pr_{v \in D}[h_2(v) \neq h_1(v) = c(v)]$$

$$x = \Pr_{v \in D}[h_1(v) = h_2(v) = c(v)]$$

$$y = \Pr_{v \in D}[h_1(v) \neq h_2(v) = c(v)]$$

$$z = \Pr_{v \in D}(h_1(v) = h_2(v) \neq c(v)]$$

Clearly,

$$w + x = \Pr_{v \in D}[h_1(v) = c(v)] = 1 - a_1, \tag{1}$$

and since c, h_1 and h_2 are Boolean,

$$y + z = \Pr_{v \in D}[h_1(v) \neq c(v)] = a_1. \tag{2}$$

In terms of these variables, we can express explicitly the chance that EX_i returns instance v:

$$D_1(v) = D(v) \tag{3}$$

$$D_2(v) = \frac{D(v)}{2}\left(\frac{p_1(v)}{a_1} + \frac{1 - p_1(v)}{1 - a_1}\right) \tag{4}$$

$$D_3(v) = \frac{D(v)q(v)}{w + y} \tag{5}$$

Equation (3) is trivial. To see that Equation (4) holds, note that the chance that the initial coin flip comes up *tails* is 1/2, and the chance that instance v is the first instance misclassified by h_1 is $D(v)p_1(v)/a_1$. The case that the coin comes up *heads* is handled in a similar fashion, as is the derivation of Equation (5).

From Equation (4), we have that

$$\begin{aligned} 1 - a_2 &= \sum_{v \in X_n} D_2(v)(1 - p_2(v)) \\ &= \frac{1}{2a_1}\sum_{v \in X_n} D(v)p_1(v)(1 - p_2(v)) + \frac{1}{2(1 - a_1)}\sum_{v \in X_n} D(v)(1 - p_1(v))(1 - p_2(v)) \\ &= \frac{y}{2a_1} + \frac{z}{2(1 - a_1)} \end{aligned} \tag{6}$$

(Note that Equation (6) could also have been derived from Figure 3 in the case of deterministic hypotheses: if β is as shown in the figure, then it is not hard to see that $y = 2a_1\beta$ and $x = 2(1 - a_1)(1 - a_2 - \beta)$. These imply Equation (6).)

Combining Equations (1), (2) and (6), we see that the values of w and z can be solved for and written explicitly in terms of y, a_1 and a_2:

$$w = (2a_2 - 1)(1 - a_1) + \frac{y(1 - a_1)}{a_1}$$

$$z = a_1 - y$$

Using these values and Equation (5), we are finally ready to compute the error of the output hypothesis h:

$$\begin{aligned}
\Pr_{v \in D}[h(v) \neq c(v)] &= \Pr_{v \in D}[(h_1(v) = h_2(v) \neq c(v)) \vee (h_1(v) \neq h_2(v) \wedge h_3(v) \neq c(v))] \\
&= z + \sum_{v \in X_n} D(v)q(v)p_3(v) \\
&= z + \sum_{v \in X_n} (w + y)D_3(v)p_3(v) \\
&= z + a_3(w + y) \\
&\leq z + \alpha(w + y) \\
&= \alpha(2a_2 - 1)(1 - a_1) + a_1 + \frac{y(\alpha - a_1)}{a_1} \\
&\leq \alpha(2a_2 - 1)(1 - a_1) + \alpha \\
&\leq \alpha(2\alpha - 1)(1 - \alpha) + \alpha = 3\alpha^2 - 2\alpha^3 = g(\alpha) = \epsilon
\end{aligned}$$

as desired. The inequalities here follow from the facts that each $a_i \leq \alpha < 1/2$, and that, by Equation (2), $y \leq a_1$.

This completes the proof. ■

3.4. Analysis

In this section, we argue that `Learn` runs in polynomial time. Here and throughout this section, unless stated otherwise, polynomial refers to polynomial in n, s, $1/\epsilon$ and $1/\delta$. Our approach will be first to derive a bound on the *expected* running time of the procedure, and then to use a part of the confidence δ to bound with high-probability the actual running

time of the algorithm. Thus, we will have shown that the procedure is probably fast and correct, completing the proof of Theorem 1. (Although technically we only show that Learn halts probabilistically, using techniques described by Haussler, Kearns, Littlestone and Warmuth (1988), the procedure can easily be converted into a learning algorithm that halts deterministically in polynomial time.)

We will be interested in bounding several quantities. First, we are of course interested in bounding the expected running time $T(\epsilon, \delta)$ of Learn(ϵ, δ, EX). This running time in turn depends on the time $U(\epsilon, \delta)$ to evaluate an hypothesis returned by Learn, and on the expected number of examples $M(\epsilon, \delta)$ needed by Learn. In addition, let $t(\delta)$, $u(\delta)$ and $m(\delta)$ be analogous quantities for WeakLearn(δ, EX). By assumption, t, u and m are polynomially bounded. Also, all of these functions depend implicitly on n and s.

As a technical point, we note that the expectations denoted by T and M are taken only over good runs of Learn. That is, the expectations are computed given the assumption that every sub-hypothesis and every estimator is successfully computed with the desired accuracy. By Theorem 2, Learn will have a good run with probability at least $1 - \delta$.

It is also important to point out that T (respectively, t) is the expected running time of Learn (WeakLearn) when called with an oracle EX that provides examples *in unit time*. Our analysis will take into account the fact that the simulated oracles supplied to Learn or WeakLearn at lower levels of the recursion do not in general run in unit time.

We will see that T, U and M are all exponential in the depth of the recursion induced by calling Learn. We therefore begin by bounding this depth. Let $B(\epsilon, p)$ be the smallest integer i for which $g^i(1/2 - 1/p) \leq \epsilon$. On each recursive call, ϵ is replaced by $g^{-1}(\epsilon)$. Thus, the depth of the recursion is bounded by $B(\epsilon, p(n, s))$. We have:

LEMMA 1. The depth of the recursion induced by calling Learn(ϵ, δ, EX) is at most $B(\epsilon, p(n, s)) = O(\log(p(n, s)) + \log\log(1/\epsilon))$.

Proof. We can say $B(\epsilon, p(n, s)) \leq b + c$ if $g^b(1/2 - 1/p(n, s)) \leq 1/4$ and $g^c(1/4) \leq \epsilon$. Clearly, $g(x) \leq 3x^2$ and so $g^i(x) \leq (3x)^{2^i}$. Thus, $g^c(1/4) \leq \epsilon$ if $c = \lceil \lg \log_{4/3}(1/\epsilon) \rceil$. Similarly, if $1/4 \leq x \leq 1/2$ then $1/2 - g(x) = (1/2 - x)(1 + 2x - 2x^2) \geq (11/8)(1/2 - x)$. This implies that $1/2 - g^i(x) \geq (11/8)^i(1/2 - x)$, assuming that $x, g(x), \ldots, g^{i-1}(x)$ are all at least 1/4. Thus, $g^b(1/2 - 1/p(n, s)) \leq 1/4$ if $b = \lceil \log_{11/8}(p(n, s)/4) \rceil$. ■

For the remainder of this analysis, we let $p = p(n, s)$ and, where clear from context, let $B = B(\epsilon, p)$. Note that $B(g^{-1}(\epsilon), p) = B - 1$ for $\epsilon < 1/2 - 1/p$.

We show next that U is polynomially bounded. This is important because we require that the returned hypothesis be polynomially evaluatable.

LEMMA 2. The time to evaluate an hypothesis returned by Learn(ϵ, δ, EX) is $U(\epsilon, \delta) = O(3^B \cdot u(\delta/5^B))$.

Proof. If $\epsilon \geq 1/2 - 1/p$, then Learn returns an hypothesis computed by WeakLearn. In this case, $U(\epsilon, \delta) = u(\delta)$. Otherwise, the hypothesis returned by Learn involves the computation of at most three sub-hypotheses. Thus,

$$U(\epsilon, \delta) \leq 3 \cdot U(g^{-1}(\epsilon), \delta/5) + c$$

for some positive constant c. A straightforward induction argument shows that this recurrence implies the bound

$$U(\epsilon, \delta) \leq 3^B u(\delta/5^B) + c(3^B - 1). \quad \blacksquare$$

When an example is requested of a simulated oracle on one of Learn's recursive calls, that oracle must itself draw several examples from its own oracle EX. For instance, on the third recursive call, the simulated oracle must draw instances until it finds one on which h_1 and h_2 disagree. Naturally, the running time of Learn depends on how many examples must be drawn in this manner by the simulated oracle. The next lemma bounds this quantity.

LEMMA 3. Let r be the expected number of examples drawn from EX by any oracle EX_i simulated by Learn on a good run when asked to provide a single example. Then $r \leq 4/\epsilon$.

Proof. When Learn calls itself the first time (to find h_1), the examples oracle EX it was passed is left unchanged. In this case, $r = 1$.

The second time Learn calls itself, the constructed oracle EX_2 loops each time it is called until it receives a desirable example. Depending on the result of the initial coin flip, we expect EX_2 to loop $1/a_1$ or $1/(1 - a_1)$ times. Note that if $a_1 \leq \epsilon - 2\tau_1 = \epsilon/3$ then, based on its estimate of a_1, Learn would have simply returned h_1 instead of making a second or third recursive call. Thus, we can assume $\epsilon/3 \leq a_1 \leq 1/2$, and so $r \leq 3/\epsilon$ in this case.

Finally, when Learn calls itself the third time, we expect the constructed oracle EX_3 to loop $1/(w + y)$ times before finding a suitable example. (Here, the variables w, x, y and z are as defined in the proof of Theorem 2.) It remains then only to show that $w + y \geq \epsilon/4$. Note that the error e of h_2 on the original distribution D is $w + z$. Thus, using this fact and Equations (1), (2) and (6), we can solve explicitly for w and y in terms of e, a_1 and a_2, and so find that

$$w + y = a_1 + \frac{e - 4a_1a_2(1 - a_1)}{1 - 2a_1} \geq a_1 + \frac{e - 4a_1\alpha(1 - a_1)}{1 - 2a_1} \tag{7}$$

Regarding e and $\alpha < 1/2$ as fixed, we will refer to this last function on the right hand side of the inequality as $f(a_1)$. To lower bound $w + y$, we will find the minimum of f on the interval $[0, \alpha]$. The derivative of f is:

$$f'(a_1) = \frac{(4 - 8\alpha)a_1^2 - (4 - 8\alpha)a_1 + (1 - 4\alpha + 2e)}{(1 - 2a_1)^2}.$$

The denominator of this derivative is clearly zero only when $a_1 = 1/2$, and the numerator, being a parabola centered about the line $a_1 = 1/2$, has at most one zero less than $1/2$. Thus, the function f has at most one critical point on the interval $(-\infty, 1/2)$. Furthermore, since f tends to $-\infty$ as $a_1 \to -\infty$, a single critical point in this range cannot possibly be

minimal. This means that f's minimum on any closed subinterval of $(-\infty, 1/2)$ is achieved at one endpoint of the subinterval. In particular, for the subinterval of interest to us, the function achieves its minimum either when $a_1 = 0$ or when $a_1 = \alpha$. Thus, $w + y \geq \min(f(0), f(\alpha))$.

We can assume that $e \geq \epsilon - 2\tau_2 = (3/4 + \alpha/2)\epsilon$; otherwise, if e were smaller than this quantity, then Learn would have returned h_2 rather than going on to compute h_3. Thus, $f(0) = e \geq 3\epsilon/4$, and, using our bound for e and the fact that $\epsilon = 3\alpha^2 - 2\alpha^3$,

$$f(\alpha) = \frac{\alpha - 6\alpha^2 + 4\alpha^3 + e}{1 - 2\alpha}$$

$$\geq \frac{\alpha - 6\alpha^2 + 4\alpha^3 + \left(\frac{3}{4} + \frac{1}{2}\alpha\right)(3\alpha^2 - 2\alpha^3)}{1 - 2\alpha}$$

$$= \frac{1}{4}\alpha(4 - 7\alpha + 2\alpha^2).$$

Since $4 - 7\alpha + 2\alpha^2 \geq 1$ for $\alpha \leq 1/2$, $f(\alpha) \geq \alpha/4 \geq \epsilon/4$. We conclude $w + y \geq \epsilon/4$, completing the proof. ■

To bound the number of examples needed to estimate a_1 and e, we will make use of the following bounds on the tails of a binomial distribution (Angluin and Valiant, 1979; Hoeffding, 1963).

LEMMA 4. (Chernoff Bounds) Consider a sequence of m independent Bernoulli trials, each succeeding with probability p. Let S be the random variable describing the total number of successes. Then for $0 \leq \gamma \leq 1$, the following hold:

- $\Pr[|S - mp| \geq \gamma m] \leq 2e^{-2m\gamma^2}$,
- $\Pr[S \leq (1 - \gamma)mp] \leq e^{-\gamma^2 mp/2}$, and
- $\Pr[S \geq (1 + \gamma)mp] \leq e^{-\gamma^2 mp/3}$.

LEMMA 5. On a good run, the expected number of examples $M(\epsilon, \delta)$ needed by Learn (ϵ, δ, EX) is

$$O\left(\frac{36^B}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right)$$

Proof. In the base case that $\epsilon \geq 1/2 - 1/p$, Learn simply calls WeakLearn, so we have $M(\epsilon, \delta) = m(\delta)$. Otherwise, on each of the recursive calls, the simulated oracle is required to provide $M(g^{-1}(\epsilon), \delta/5)$ examples. To provide one such example, the simulated oracle must itself draw at most an average of $4/\epsilon$ examples from EX. Thus, each recursive call demands at most $(4/\epsilon) \cdot M(g^{-1}(\epsilon), \delta/5)$ examples on average.

In addition, Learn requires some examples for making its estimates $\hat{a}_1$ and $\hat{e}$. Using the first bound of Lemma 4, it follows that a sample of size $O(\log(1/\delta)/\tau_i^2)$ suffices for each estimate. Note that $1/p \leq 1/2 - \epsilon = 1/2 - g(\alpha) = (1/2 - \alpha)(1 + 2\alpha - 2\alpha^2) \leq (3/2)(1/2 - \alpha)$. Thus, by our choice of τ_1 and τ_2, both estimates can be made using $O(p^2 \log(1/\delta)/\epsilon^2)$ examples.

We thus arrive at the recurrent inequality:

$$M(\epsilon, \delta) \leq \frac{12}{\epsilon} \cdot M(g^{-1}(\epsilon), \delta/5) + \frac{cp^2 \log(1/\delta)}{\epsilon^2} \tag{8}$$

for some positive constant c. To complete the proof, we argue inductively that this implies the bound

$$M(\epsilon, \delta) \leq \frac{36^B \cdot m(\delta/5^B) + c(36^B - 1)p^2 \log(5^B/\delta)}{\epsilon^2}. \tag{9}$$

The base case clearly satisfies this bound. In the general case, Equation (8) implies by inductive hypothesis that

$$\begin{aligned} M(\epsilon, \delta) &\leq \frac{12}{\epsilon} \cdot \left[\frac{36^{B-1} \cdot m(\delta/5^B) + c(36^{B-1} - 1)p^2 \log(5^B/\delta)}{(g^{-1}(\epsilon))^2}\right] + \frac{cp^2 \log(1/\delta)}{\epsilon^2} \\ &\leq \frac{12}{\epsilon} \cdot \left[\frac{36^{B-1} \cdot m(\delta/5^B) + c(36^{B-1} - 1)p^2 \log(5^B/\delta)}{\epsilon/3}\right] + \frac{cp^2 \log(1/\delta)}{\epsilon^2} \\ &= \frac{36^B \cdot m(\delta/5^B) + c(36^B - 1)p^2 \log(5^B/\delta)}{\epsilon^2} - \frac{cp^2}{\epsilon^2}(35 \log(1/\delta) + 36 \log 5) \end{aligned}$$

which clearly implies Equation (9). The last inequality here follows from the fact that $\epsilon \leq 3(g^{-1}(\epsilon))^2$ since $g(\alpha) \leq 3\alpha^2$ for $\alpha \geq 0$. ■

LEMMA 6. On a good run, the expected execution time of Learn(ϵ, δ, EX) is given by

$$T(\epsilon, \delta) = O\left[3^B \cdot t(\delta/5^B) + \frac{108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right].$$

Proof. As in the previous lemmas, the base case that $\epsilon \geq 1/2 - 1/p$ is easily handled. In this case, $T(\epsilon, \delta) = t(\delta)$.

Otherwise, Learn takes time $3 \cdot T(g^{-1}(\epsilon), \delta/5)$ on its three recursive calls. In addition, Learn spends time drawing examples to make the estimates $\hat{a}_1$ and $\hat{e}$, and overhead time is also spent by the simulated examples oracles passed on the three recursive calls. A typical example that is drawn from Learn's oracle EX is evaluated on zero, one or two of the previously computed sub-hypotheses. For instance, an example drawn for the purpose of estimating $\hat{a}_1$ is evaluated once by h_1; an example drawn for the simulated oracle EX_3 is evaluated by both h_1 and h_2. Thus, Learn's overhead time is proportional to the product

of the total number of examples needed by Learn and the time it takes to evaluate a sub-hypothesis on one of these examples. Therefore, the following recurrence holds:

$$T(\epsilon, \delta) \leq 3 \cdot T(g^{-1}(\epsilon), \delta/5) + O(U(g^{-1}(\epsilon), \delta/5) \cdot M(\epsilon, \delta)) \tag{10}$$

Applying Lemmas 2 and 5, this implies

$$T(\epsilon, \delta) \leq 3 \cdot T(g^{-1}(\epsilon), \delta/5) + \frac{c \cdot 108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))$$

for some positive constant c. A straightforward induction argument shows that this implies:

$$T(\epsilon, \delta) \leq 3^B \cdot t(\delta/5^B) + \frac{2c \cdot 108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B)). \quad \blacksquare$$

The main result of this section follows immediately:

THEOREM 3. Let $0 < \epsilon < 1/2$ and let $0 < \delta \leq 1$. With probability at least $1 - \delta$, the execution of Learn$(\epsilon, \delta/2, EX)$ halts in polynomial time and outputs an hypothesis ϵ-close to the target concept.

Proof. By Theorem 2, the chance that Learn does not have a good run is at most $\delta/2$. By Markov's inequality and Lemma 6, the chance that Learn on a good run fails to halt in time $(2/\delta) \cdot T(\epsilon, \delta/2)$ is also at most $\delta/2$. Thus, the probability that Learn has a good run (and so outputs an ϵ-close hypothesis) and halts in polynomial time is at least $1 - \delta$. ■

3.5. Space complexity

Although not of immediate consequence to the proof of Theorem 3, it is worth pointing out that Learn's space requirements are relatively modest, as proved in this section.

Let $S(\epsilon, \delta)$ be the space used by Learn(ϵ, δ, EX); let $Q(\epsilon, \delta)$ be the space needed to store an output hypothesis; and let $R(\epsilon, \delta)$ be the space needed to evaluate such an hypothesis. Let $s(\delta)$, $q(\delta)$ and $r(\delta)$ be analogous quantities for WeakLearn(δ, EX). Then we have:

LEMMA 7. The space $Q(\epsilon, \delta)$ required to store an hypothesis output by Learn(ϵ, δ, EX) is at most $O(3^B \cdot q(\delta/5^B))$. The space $R(\epsilon, \delta)$ needed to evaluate such an hypothesis is $O(B + r(\delta/5^B))$. Finally, the total space $S(\epsilon, \delta)$ required by Learn is $O(3^B \cdot q(\delta/5^B) + s(\delta/5^B) + B \cdot r(\delta/5^B))$.

Proof. For $\epsilon \geq 1/2 - 1/p$, the bounds are trivial. To bound Q, note that the hypothesis returned by Learn is a composite of three (or fewer) hypotheses. Thus,

$$Q(\epsilon, \delta) \leq 3 \cdot Q(g^{-1}(\epsilon), \delta/5) + O(1).$$

To evaluate such a composite hypothesis, each of the sub-hypotheses is evaluated one at a time. Thus,

$$R(\epsilon, \delta) \leq R(g^{-1}(\epsilon), \delta/5) + O(1).$$

Finally, to bound S, note that the space required by Learn is dominated by the storage of the sub-hypotheses, by their recursive computation, and by the space needed to evaluate them. Since the sub-hypotheses are computed one at a time, we have:

$$S(\epsilon, \delta) \leq S(g^{-1}(\epsilon), \delta/5) + O(Q(g^{-1}(\epsilon), \delta/5) + R(g^{-1}(\epsilon), \delta/5)).$$

The solutions of these three recurrences are all straightforward, and imply the stated bounds. ■

4. Improving Learn's time and sample complexity

In this section, we describe a modification to the construction of Section 3 that significantly improves Learn's time and sample complexity. In particular, we will improve these complexity measures by roughly a factor of $1/\epsilon$, giving bounds that are linear in $1/\epsilon$ (ignoring log factors). These improved bounds will have some interesting consequences, described in later sections.

In the original construction of Learn, much time and many examples are squandered by the simulated oracles EX_i waiting for a desirable instance to be drawn. Lemma 3 showed that the expected time spent waiting is $O(1/\epsilon)$. The modification described below will reduce this to $O(1/\alpha) = O(1/\sqrt{\epsilon})$. (Here, $\alpha = g^{-1}(\epsilon)$ as before.)

Recall that the running time of oracle EX_2 depends on the error a_1 of the first sub-hypothesis h_1. In the original construction, we ensured that a_1 not be too small by estimating its value, and, if smaller than ϵ, returning h_1 instead of continuing the normal execution of the subroutine. Since this approach only guarantees that $a_1 \geq \Omega(\epsilon)$, there does not seem to be any way of ensuring that EX_2 run for $o(1/\epsilon)$ time. To improve EX_2's running time then, we will instead modify h_1 by deliberately *increasing* its error. Ironically, this intentional injection of error will have the effect of improving Learn's worst case running time by limiting the time spent by either EX_2 or EX_3 waiting for a suitable instance.

4.1. The modifications

Specifically, here is how Learn is modified. Call the new procedure Learn′. Following the recursive computation of h_1, Learn′ estimates the error a_1 of h_1, although less accurately than Learn. Let $\hat{a}_1$ be this estimate, and choose a sample large enough that $|a_1 - \hat{a}_1| \leq \alpha/4$ with probability at least $1 - \delta/5$. Since $0 \leq a_1 \leq \alpha$, we can assume without loss of generality that $\alpha/4 \leq \hat{a}_1 \leq 3\alpha/4$.

Next, Learn′ defines a new hypothesis h_1' as follows: given an instance v, h_1' first flips a coin biased to turn up *heads* with probability exactly

$$\beta = \frac{\frac{3}{4}\alpha - \hat{a}_1}{1 - \frac{1}{4}\alpha - \hat{a}_1}$$

If the outcome is *tails*, then h_1' evaluates $h_1(v)$ and returns the result. Otherwise, if *heads*, h_1' predicts the wrong answer, $\neg c(v)$. Since h_1' will only be used during the training phase, we can assume that the correct classification of v is available, and thus that h_1' can be simulated.

This new hypothesis h_1' is now used in place of h_1 by EX_2 and EX_3. The rest of the subroutine is unmodified (aside from one other minor modification described in Lemma 8 below). In particular, the final returned hypothesis h is unchanged—that is, h_1, not h_1', is used by h.

4.2. Correctness

To see that Learn' is correct, we will assume as in the proof of Theorem 2 that a good run occurs; this will be the case with probability at least $1 - \delta$. Note first that the error of h_1' is exactly $a_1' = (1 - \beta)a_1 + \beta$ since the chance of error is a_1 on *tails*, and is 1 on *heads*. By our choice of β, it can be verified that $\alpha/2 \leq a_1' \leq \alpha$.

Let h' be the same hypothesis as h, except with h_1' used in lieu of h_1. Note that h', h_1', h_2 and h_3 are related to one another in exactly the same way that h, h_1, h_2 and h_3 are related in the original proof of Theorem 2. That is, if we imagine that h_1' is returned on the first recursive call of the original procedure Learn, then it is not impossible that h_2 and h_3 would be returned on the second and third recursive calls, in which case h' would be the returned hypothesis. Put another way, the proof that h' has error at most $g(\alpha) = \epsilon$ is an identical copy of the one given in the proof of Theorem 2, except that all occurrences of h and h_1 are replaced by h' and h_1'.

Finally, we must shown that h's error is at most that of h'. Let $p_i'(v) = \Pr[h_i'(v) \neq c(v)]$, and let $p_i(v)$ be as in Theorem 2. Then for $v \in X_n$, we have

$$\begin{aligned}
\Pr[h'(v) \neq c(v)] &= p_1'(v)[(1 - p_2(v))p_3(v) + p_2(v)(1 - p_3(v))] + p_2(v)p_3(v) \\
&\geq p_1(v)[(1 - p_2(v))p_3(v) + p_2(v)(1 - p_3(v))] + p_2(v)p_3(v) \\
&= \Pr[h(v) \neq c(v)].
\end{aligned}$$

where the inequality follows from the observation that $p_1'(v) = (1 - \beta)p_1(v) + \beta \geq p_1(v)$. This implies that the error of h is at most the error of h', which is bounded by ϵ.

4.3. Analysis

Next, we show that Learn' runs faster using fewer examples than Learn. We use essentially the same analysis as in Section 3.4. The following three lemmas are modified versions of Lemmas 3, 5 and 6. The proofs of the other lemmas apply immediately to Learn' with little or no modification, and so are ommitted.

LEMMA 8. Let r be the expected number of examples drawn from EX by any oracle EX_i simulated by Learn′ on a good run when asked to provide a single example. Then $r \leq 4/\alpha$.

Proof. As in the original proof, $r = 1$ for EX_1. We expect the second oracle to loop at most $1/a_1'$ times on average. Since $a_1' \geq \alpha/2$, r is at most $2/\alpha$ in this case.

Finally, to bound the number of iterations of EX_3, we will shown that $w + y \geq \alpha/4$ using Equation (7) as in the original proof. To lower bound $w + y$, we find the minimum of the last formula f of Equation (7) (with a_1 replaced by a_1' of course) on the interval $[\alpha/2, \alpha]$. As noted previously, the function f must achieve its minimum at one endpoint of the interval. We assume as in the original proof that $e \geq (3/4 + \alpha/2)\epsilon$. It was previously shown that $f(\alpha) \geq \alpha/4$, and, by a similar argument, we can show $f(\alpha/2) \geq \alpha/2 + \alpha^3 + \alpha^2/4$ $(1 - \alpha) \geq \alpha/2$. This completes the proof. ■

LEMMA 9. On a good run, the expected number of examples $M(\epsilon, \delta)$ needed by Learn′ (ϵ, δ, EX) is

$$O\left(\frac{36^B}{\epsilon} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right).$$

Proof. The proof is nearly the same as for Lemma 5. In addition to incorporating the superior bound given by Lemma 8 on the number of examples needed by the simulated oracles, we must also consider the number of examples needed to estimate a_1 and e. The first, a_1, can be estimated using a sample of size $O(\log(1/\delta)/\alpha^2) = O(\log(1/\delta)/\epsilon)$; this can be derived from the first bound of Lemma 4, and by noting that $\epsilon = g(\alpha) \leq 3\alpha^2$ for $\alpha \geq 0$. By estimating e in a slightly different manner, we can also achieve a better bound on the sample size needed. Specifically, we can choose a sample large enough that, with probability $1 - \delta/5$, $\hat{e} \leq \epsilon - \tau_2$ if $e \leq \epsilon - 2\tau_2$, and $\hat{e} \geq \epsilon - \tau_2$ if $e \geq \epsilon$. Such an estimate has all of the properties needed by Learn′, but only requires a sample of size $O(p^2 \log(1/\delta)/\epsilon)$ as can be derived using the second and third bound of Lemma 4. (See Haussler, Kearns, Littlestone and Warmuth (1988) for a detailed example of this sort of calculation.)

Thus, we arrive at the recurrence

$$M(\epsilon, \delta) \leq \frac{12}{g^{-1}(\epsilon)} \cdot M(g^{-1}(\epsilon), \delta/5) + O\left(\frac{p^2 \log(1/\delta)}{\epsilon}\right)$$

which implies the stated bound by an argument similar to that given in the proof of Lemma 5. ■

LEMMA 10. On a good run, the expected execution time of Learn′(ϵ, δ, EX) is given by

$$T(\epsilon, \delta) = O\left(3^B \cdot t(\delta/5^B) + \frac{108^B \cdot u(\delta/5^B)}{\epsilon} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right).$$

Proof. This bound follows from Equation (10), using the superior bound on M given by Lemma 9. ■

5. Variations on the learning model

Next, we consider how the main result relates to some other learning models.

5.1. Group learning

An immediate consequence of Theorem 1 concerns *group learnability*. In the group learning model, the learner produces a hypothesis that need only correctly classify large groups of instances, all of which are either positive or negative examples. Kearns and Valiant (1989) prove the equivalence of group learning and weak learning. Thus, by Theorem 1, group learning is also equivalent to strong learning.

5.2. Miscellaneous PAC models

Haussler, Kearns, Littlestone and Warmuth (1988) describe numerous variations on the basic PAC model, and show that all of them are equivalent. For instance, they consider randomized versus deterministic algorithms, algorithms for which the size s of the target concept is known or unknown, and so on. It is not hard to see that all of their equivalence proofs apply to weak learning algorithms as well (with one exception described below), and so that any of these weak learning models are equivalent by Theorem 1 to the basic PAC-learning model.

The one reduction from their paper that does not hold for weak learning algorithms concerns the equivalence of the one- and two-oracle learning models. In the one-oracle model (used exclusively in this paper), the learner has access to a single source of positive and negative examples. In the two-oracle model, the learner has access to one oracle that returns only positive examples, and another returning only negative examples. The authors show that these models are equivalent for strong learning algorithms. However, their proof apparently cannot be adapted to show that one-oracle weak learnability implies two-oracle weak learnability (although their proof of the converse is easily and validly adapted). This is because their proof assumes that the error ϵ can be made arbitrarily small, clearly a bad assumption for weak learning algorithms. Nevertheless, this is not a problem since we have shown that one-oracle weak learnability implies one-oracle strong learnability, which in turn implies two-oracle strong (and therefore weak) learnability. Thus, despite the inapplicability of Haussler et al.'s original proof, all four learning models are equivalent.

5.3. Fixed hypotheses

Much of the PAC-learning research has been concerned with the form or representation of the hypotheses output by the learning algorithm. Clearly, the construction described in Section 3 does not in general preserve the form of the hypotheses used by the weak learning algorithm. It is natural to ask whether there exists any construction preserving this form. That is, if concept class $\boldsymbol{C}$ is weakly learnable by an algorithm using hypotheses

from a class H of representations, does there then exist a strong learning algorithm for C that also only outputs hypotheses from H?

In general, the answer to this question is *no* (modulo some relatively weak complexity assumptions). As a simple example, consider the problem of learning k-term DNF formulas using only hypotheses represented by k-term DNF. (A formula in disjunctive normal form (DNF) is one written as a disjunction of terms, each of which is a conjunction of literals, a literal being either a variable or its complement.) Pitt and Valiant (1988) show that this learning problem is infeasible if RP $\neq$ NP for k as small as 2.

Nevertheless, the weak learning problem is solved by the algorithm sketched below. (A similar algorithm is given by Kearns (1988).) First, choose a "large" sample. If significantly more than half of the examples in the sample are negative (positive), then output the "always predict negative (positive)" hypothesis, and halt. Otherwise, we can assume that the distribution is roughly evenly split between positive and negative examples. Select and output the disjunction of k or fewer literals that misclassifies none of the positive examples, and the fewest of the negative examples.

We briefly argue that this hypothesis is, with high probability, $(1/2 - \Omega(1/n^k))$-close to the target concept. First, note that the target k-term DNF formula is equivalent to some k-CNF formula (Pitt and Valiant, 1988). (A formula in conjunctive normal form (CNF) is one written as the conjunction of clauses, each clause a disjunction of literals. If each clause consists of only k literals, then the formula is in k-CNF.) Next, we observe that every clause is satisfied by every assignment that satisfies the entire k-CNF formula. Moreover, since the formula has at most $O(n^k)$ clauses, by an averaging argument, there must be one clause not satisfied by $\Omega(1/n^k)$ of the assignments (as weighted by the target distribution) that do not satisfy the entire formula. Thus, there exists some disjunction of k literals that is correct for nearly all of the positive examples and for at least $\Omega(1/n^k)$ of the negative examples. In particular, the output hypothesis has this property. Since the distribution is roughly evenly divided between positive and negative examples, this implies that the output hypothesis is roughly $(1/2 - \Omega(1/n^k))$-close to the target formula.

5.4. Queries

A number of researchers have considered learning scenarios in which the learner is not only able to passively observe randomly selected examples, but is also able to ask a "teacher" various sorts of questions or *queries* about the target concept. For instance, the learner might be allowed to ask if some particular instance is a positive or negative example. Angluin (1988) describes several kinds of query that might be useful to the learner. The purpose of this section is simply to point out that the construction of Section 3 is applicable even in the presence of most kinds of query. That is, a weak learning algorithm that depends on the availability of certain kinds of query can be converted, using the same construction, into a strong learning algorithm using the same query types.

5.5. Many-valued concepts

In this paper, we have only considered Boolean valued concepts, that is, concepts that classify every instance as either a positive or a negative example. Of course, in the "real world,"

most learning tasks require classification into one of several categories (for instance, character recognition). How does the result generalize to handle many-valued concepts?

First of all, for learning a k-valued concept, it is not immediately clear how to define the notion of weak learnability. An hypothesis that guesses randomly on every instance will be correct only $1/k$ of the time, so one natural definition would require only that the weak learning algorithm classify instances correctly slightly more than $1/k$ of the time. Unfortunately, under this definition, strong and weak learnability are inequivalent for k as small as three. As an informal example, consider learning a concept taking the values 0, 1 and 2, and suppose that it is "easy" to predict when the concept has the value 2, but "hard" to predict whether the concept's value is 0 or 1. Then to weakly learn such a concept, it suffices to find an hypothesis that is correct whenever the concept is 2, and that guesses randomly otherwise. For any distribution, this hypothesis will be correct half of the time, achieving the weak learning criterion of accuracy significantly better than 1/3. However, boosting the accuracy further is clearly infeasible.

Thus, a better definition of weak learnability is one requiring that the hypothesis be correct on slightly more than half of the distribution, regardless of k. Using this definition, the construction of Section 3 is easily modified to handle many-valued concepts.

6. General complexity bounds for PAC learning

The construction derived in Sections 3 and 4 yields some unexpected relationships between the allowed error ϵ and various complexity measures that might be applied to a strong learning algorithm. One of the more surprising of these is a proof that, for every learnable concept class, there exists an efficient algorithm whose output hypotheses can be evaluated in time polynomial in $\log(1/\epsilon)$. Furthermore, such an algorithm's space requirements are also only poly-logarithmic in $1/\epsilon$—far less, for instance, than would be needed to store the entire sample. In addition, its time and sample size requirements grow only linearly in $1/\epsilon$ (disregarding log factors).

THEOREM 4. If $\boldsymbol{C}$ is a learnable concept class, then there exists an efficient learning algorithm for $\boldsymbol{C}$ that:

- requires a sample of size $\dfrac{p_1(n, s, \log(1/\epsilon), \log(1/\delta))}{\epsilon}$,
- halts in time $\dfrac{p_2(n, s, \log(1/\epsilon), \log(1/\delta))}{\epsilon}$,
- uses space $p_3(n, s, \log(1/\epsilon), \log(1/\delta))$, and
- outputs hypotheses of size $p_4(n, s, \log(1/\epsilon))$, evaluatable in time $p_5(n, s, \log(1/\epsilon))$

for some polynomials p_1, p_2, p_3, p_4 and p_5.

Proof. Given a strong learning algorithm A for $\boldsymbol{C}$, "hard-wire" $\epsilon = 1/4$, thus converting A into a weak learning algorithm A' that outputs hypotheses 1/4-close to the target concept.

Now let A'' be the procedure obtained by applying the construction of `Learn'` with A' plugged in for `WeakLearn`. As remarked previously, we can assume without loss of generality that A'' halts deterministically in polynomial time. Note, by the lemmas of Sections 3 and 4 that A'' "almost" achieves the resource bounds given in the theorem, the only problem being that the bounds attained are polynomial in $1/\delta$ rather than $\log(1/\delta)$ as desired.

This problem is alleviated by applying the construction of Haussler, Kearns, Littlestone and Warmuth (1988) for converting any learning algorithm B into one running in time polynomial in $\log(1/\delta)$. Essentially, this construction works as follows: Given inputs n, s, ϵ and δ, first simulate B $O(\log 1/\delta))$ times, each time setting B's accuracy parameter to $\epsilon/4$ and B's confidence parameter to 1/2. Save all of the computed hypotheses. Next, draw a sample of $O(\log(1/\delta)/\epsilon)$ examples, and output the one that misclassifies the fewest examples in the sample. Haussler, et al. argue that the resulting procedure outputs an ϵ-close hypothesis with probability $1 - \delta$.

Applying this construction to A'', we obtain a final procedure that one can verify achieves all of the stated bounds. ■

The remainder of this section is a discussion of some of the consequences of Theorem 4.

6.1. Improving the performance of known algorithms

These bounds can be applied immediately to a number of existing learning algorithms, yielding improvements in time and/or space complexity (at least in terms of ϵ). For instance, the computation time of Blumer, Ehrenfeucht, Haussler and Warmuth's (1989) algorithm for learning half-spaces of R^n, which involves the solution of a linear programming problem of size proportional to the sample, can be improved by a polynomial factor of $1/\epsilon$. The same is also true of Baum's (1989) algorithm for learning unions of half-spaces, which involves finding the convex hull of a significant fraction of the sample.

There are many more algorithms for which the theorem implies improved space efficiency. This is especially true of the many known PAC algorithms that work by choosing a large sample and then finding an hypothesis consistent with it. For instance, this is how Rivest's (1987) decision list algorithm works, as do most of the algorithms described by Blumer, et al., as well as Helmbold, Sloan and Warmuth's (1990) construction for learning nested differences of learnable concepts. Since the entire sample must be stored, these algorithms are not terribly space efficient, and so can be dramatically improved by applying Theorem 4. Of course, these improvements typically come at the cost of requiring a somewhat larger sample (by a polynomial factor of $\log(1/\epsilon)$). Thus, there appears to be a tradeoff between space and sample size (or time) complexity.

6.2. Data compression

Blumer, Ehrenfeucht, Haussler and Warmuth (1987; 1989) have considered the relationship between learning and data compression. They have shown that, if any sample can be *compressed*—that is, represented by a prediction rule significantly smaller than the original sample—then this compression algorithm can be converted into a PAC-learning algorithm.

In some sense, the bound given in Theorem 4 on the size of the output hypothesis implies the converse. In particular, suppose $\boldsymbol{C}_n$ is a learnable concept class and that we have been given m examples $(v_1, c(v_1)), (v_2, c(v_2)), \ldots, (v_m, c(v_m))$ where each $v_i \in X_n$ and c is a concept in $\boldsymbol{C}_n$ of size s. These examples need not have been chosen at random. The data compression problem is to find a small representation for the data, that is, an hypothesis h that is significantly smaller than the original data set with the property that $h(v_i) = c(v_i)$ for each v_i. An hypothesis with this last property is said to be *consistent* with the sample.

Theorem 4 implies the existence of an efficient algorithm that outputs consistent hypotheses only poly-logarithmic in the size m of the sample. This is proved by the following theorem:

THEOREM 5. Let $\boldsymbol{C}$ be a learnable concept class. Then there exists an efficient algorithm that, given $0 < \delta \leq 1$ and m (distinct) examples of a concept $c \in \boldsymbol{C}_n$ of size s, outputs with probability at least $1 - \delta$ a deterministic hypothesis consistent with the sample and of size polynomial in n, s and $\log m$.

Proof. Pitt and Valiant (1988) show how to convert any learning algorithm into one that finds hypotheses consistent with a set of data points. The idea is to choose $\epsilon < 1/m$ and to run the learning algorithm on a (simulated) uniform distribution over the data set. Since ϵ is less than the weight placed on any element of the sample, the output hypothesis must have error zero. Applying this technique to a learning algorithm A satisfying the conditions of Theorem 4, we see that the output hypothesis has size only polynomial in n, s and $\log m$, and so is far smaller than the original sample for large m.

Technically, this technique requires that the learning algorithm output deterministic hypotheses. However, probabilistic hypotheses can also be handled by choosing a somewhat smaller value for ϵ, and by "hard-wiring" the computed probabilistic hypothesis with a sequence of random bits. More precisely, set $\epsilon = 1/2m$, and run A over the same distribution as before. Assume A has a good run. Note that the output hypothesis h can be regarded as a deterministic function of an instance v and a sequence of random bits r. Let p be the chance that, for a randomly chosen sequence r, $h(\cdot, r)$ misclassifies one or more of the instances in the sample. For such an r, the chance is certainly at least $1/m$ that an instance v is chosen (according to the simulated uniform distribution on the sample) for which $h(v, r) \neq c(v)$. Thus, the error of h is at least p/m. By our choice of ϵ, this implies that $p \leq 1/2$, or, in other words, that the probability that a random sequence r is chosen for which $h(\cdot, r)$ correctly classifies all of the m examples is at least 1/2. Thus, choosing and testing random sequences r, we can quickly find one for which the deterministic hypothesis $h(\cdot, r)$ is consistent with the sample. Finally, note that the size of this output hard-wired hypothesis is bounded by $|h| + |r|$, and that $|r|$ is bounded by the time it takes to evaluate h, which is poly-logarithmic in m. ■

Naturally, the notion of size in the preceding theorem depends on the underlying model of computation, which we have left unspecified. However, the theorem has some immediate corollaries when the learning problem is *discrete*, that is, when every instance in the domain X_n is encoded using a finite alphabet by a string of legnth bounded by a polynomial in n, and every concept in $\boldsymbol{C}$ of size s is also encoded using a finite alphabet by a string of length bounded by a polynomial in s.

COROLLARY 1. Let C be a learnable discrete concept class. Then there exists an efficient algorithm that, given $0 < \delta \leq 1$ and a sample as in Theorem 5, outputs with probability at least $1 - \delta$ a deterministic consistent hypothesis of size polynomial in n and s, and independent of m.

Proof. Since we assume (without loss of generality) that all the points of the sample are distinct, its size m cananot exceed $|X_n|$. Since $\log |X_n|$ is bounded by a polynomial in n, the corollary follows immediately. ■

Applying *Occam's Razor* of Blumer, et al. (1987), this implies the following strong general bound on the sample size needed to efficiently learn C. Although the bound is better than that given by Theorem 4 (at least in terms of ϵ), it should be pointed out that this improvement requires the sacrifice of space efficiency since the entire sample must be stored.

THEOREM 6. Let C be a learnable discrete concept class. Then there exists an efficient learning algorithm for C requiring a sample of size

$$O\left(\frac{p(n, s) + \log(1/\delta)}{\epsilon}\right)$$

for some polynomial p.

Proof. Blumer, et al. (1987) describe a technique for converting a so-called *Occam* algorithm A with the property described in Corollary 1 into an efficient learning algorithm with the stated sample complexity bound. Essentially, to make this conversion, one simply draws a sample of the stated size (choosing p appropriately), and runs A on the sample to find a consistent hypothesis. The authors argue that the computed hypothesis, simply by virtue of its small size and consistency with the sample, will be ϵ-close to the target concept with high probability. (Technically, their approach needs some minor modifications to handle, for instance, a randomized Occam algorithm; these modifications are straightforward.) ■

6.3. Hard functions are hard to learn

Theorem 4's bound on the size of the output hypothesis also implies that any hard-to-evaluate concept class is unlearnable. Although this result does not sound surprising, it was previously unclear how it might be proved: since a learning algorithm's hypotheses are technically permitted to grow polynomially in $1/\epsilon$, the learnability of such classes did not seem out of the question.

This result yields the first representation-independent hardness results not based on cryptographic assumptions. For instance, assuming P/poly $\neq$ NP/poly, the class of polynomial-size, nondeterministic Boolean circuits is not learnable. (The set P/poly (NP/poly) consists of those languages accepted by a family of polynomial-size deterministic (nondeterministic) circuits.) Furthermore, since learning pattern languages was recently shown (Schapire, 1989) to be as hard as learning NP/poly, this result shows that pattern languages are also unlearnable under this relatively weak structural assumption.

THEOREM 7. Suppose $\boldsymbol{C}$ is learnable, and assume that $X_n = \{0, 1\}^n$. Then there exists a polynomial p such that for all concepts $c \in \boldsymbol{C}_n$ of size s, there exists a circuit of size $p(n, s)$ exactly computing c.

Proof. Consider the set of 2^n pairs $\{(v, c(v)) \mid v \in X_n\}$. By Corollary 1, there exists an algorithm that, with positive probability, will output an hypothesis consistent with this set of elements of size only polynomial in n and s. Since this hypothesis is polynomially evaluatable, it can be converted using standard techniques into a circuit of the required size. ■

6.4. Hard functions are hard to approximate

By a similar argument, the bound on hypothesis size implies that any function not computable by small circuits cannot even be weakly approximated by a family of small circuits, for some distribution on the inputs.

Let f be a Boolean function on $\{0, 1\}^*$, D a distribution on $\{0, 1\}^n$ and C a circuit on n variables. Then C is said to *β-approximate f* under D if the probability is at most β that $C(v) \neq f(v)$ on an assignment v chosen randomly from $\{0, 1\}^n$ according to D.

THEOREM 8. Suppose some function f cannot be computed by any family of polynomial-size circuits. Then there exists a family of distributions $D_1, D_2, \ldots$, where D_n is over the set $\{0, 1\}^n$, such that for all polynomials p and q, there exist infinitely many n for which there exists no n-variable circuit of size at most $q(n)$ that $(1/2 - 1/p(n))$-approximates f under D_n.

Proof. Throughout this proof, we will assume without loss of generality that $p(n) = q(n) = n^k$ for some integer $k \geq 1$.

Suppose first that there exists some k such that for all n and every distribution D on $\{0, 1\}^n$, there exists a circuit of size at most n^k that $(1/2 - 1/n^k)$-approximates f under D. Then f can, in a sense, be weakly learned. More precisely, there exists an (exponential-time) procedure that, by searching exhaustively the set of all circuits of size n^k, will find one that $(1/2 - 1/n^k)$-approximates f under some given distribution D. Therefore, by Theorem 1, f is strongly learnable in a similar sense in exponential time. Applying Theorem 7 (whose validity depends only on the size of the output hypothesis, and not on the running time), this implies that f can be exactly computed by a family of polynomial-size circuits, contradicting the theorem's hypothesis.

Thus, for all $k \geq 1$, there exists an integer n and a distribution D on $\{0, 1\}^n$ such that no circuit of size at most n^k is able to $(1/2 - 1/n^k)$-approximate f under D. To complete the proof, it suffices to show that this implies the theorem's conclusion.

Let $\boldsymbol{D}_n^k$ be the set of distributions D on $\{0, 1\}^n$ for which no circuit of size at most n^k $(1/2 - 1/n^k)$-approximates f under D. It is easy to verify that $\boldsymbol{D}_n^k \supseteq \boldsymbol{D}_n^{K+1}$ for all k, n. Also, since every function can be computed by exponential size circuits, there must exist a constant $c > 0$ for which $\boldsymbol{D}_n^{c^n} = \emptyset$ for all n. Let $n[k]$ be the smallest n for which $\boldsymbol{D}_n^k \neq \emptyset$. By the preceding argument, $n[k]$ must exist. Furthermore, $n[k] \geq k/c$, which implies that the set $N = \{n[k] \mid k \geq 1\}$ cannot have finite cardinality.

To eliminate repeated elements from N, let $k_1 < k_2 < \ldots$ be such that $n[k_i] \neq n[k_j]$ for $i \neq j$, and such that $\{n[k_i] \mid i \geq 1\} = N$. Let D_i be defined as follows: if $i = n[k_j]$ for some j, then let D_i be any distribution in $\boldsymbol{D}^{k_j}_i$ (which cannot be empty by our definition of $n[k]$); otherwise, if $i \notin N$, then define D_i arbitrarily. Then $D_1, D_2, \ldots$ is the desired family of "hard" distributions. For if k is any integer, then for all $k_i \geq k$, $D_{n[k_i]} \in \boldsymbol{D}^{k_i}_{n[k_i]} \subseteq \boldsymbol{D}^{k}_{n[k_i]}$. This proves the theorem. ■

Informally, Theorem 8 states that any language not in the complexity class P/poly cannot be even weakly approximated by any other language in P/poly under some "hard" family of distributions. In fact, the theorem can easily be modified to apply to other circuit classes as well, including monotone P/poly, and monotone or non-monotone NC^k for fixed k. (The class NC^k consists of all languages accepted by polynomial-size circuits of depth at most $O(\log^k n)$, and a monotone circuit is one in which no negated variables appear.) In general, the theorem applies to all circuit classes closed under the transformation on hypotheses resulting from the construction of Section 3 and 4.

6.5. On-line learning

Finally, we consider implications of Theorem 4 for on-line learning algorithms. In the *on-line* learning model, the learner is presented one (randomly selected) instance at a time in a series of *trials*. Before being told its correct classification, the learner must try to predict whether the instance is a positive or negative example. An incorrect prediction is called a *mistake*. In this model, the learner's goal is to minimize the number of mistakes.

Previously, Haussler, Littlestone and Warmuth (1987) have shown that a concept class $\boldsymbol{C}$ is learnable if and only if there exists an on-line learning algorithm for $\boldsymbol{C}$ with the properties that:

- the probability of a mistake on the m-th trial is at worst linear in $m^{-\beta}$ for some constant $0 < \beta \leq 1$, and (equivalently)
- the expected number of mistakes on the first m trials is at worst linear in m^{α} for some constant $0 \leq \alpha < 1$.

(This result is also described in their paper with Kearns (1988).) Noting several examples of learning algorithms for which this second bound only grows poly-logarithmically in m, the authors ask if *every* learnable concept class has an algorithm attaining such a bound. Theorem 8 below answers this open question affirmatively, showing that in general the expected number of mistakes on the first m trials need only grow as a polynomial in $\log m$. Thus, we expect only a minute fraction of the first m predictions to be incorrect.

(This result should not be confused with those presented in another paper by Haussler, Littlestone and Warmuth (1988). In this paper, the authors describe a general algorithm applicable to a wide collection of concept classes, and they show that the expected number of mistakes made by this algorithm on the first m trials is linear in $\log m$. However, their algorithm requires exponential computation time, even if it is known that the concept class

is learnable. In contrast, Theorem 8 states that, if a concept class is learnable, then there exists an efficient algorithm making poly-logarithmic in m mistakes on average on the first m trials.)

Haussler, Littlestone and Warmuth (1987) also consider the space efficiency of on-line learning algorithms. They define a *space-efficient* learning algorithm to be one whose space requirements on the first m trials do not exceed a polynomial in n, s and $\log m$. Thus, a space-efficient algorithm is one using far less memory than would be required to store explicitly all of the preceding observations. The authors describe a number of space-efficient algorithms (though are unable to find one for learning unions of axis-parallel rectangles in the plane), and so are led to ask whether there exist space-efficient algorithms for *all* learnable concept classes. Surprisingly, this open question can also be answered afirmatively, as proved by the theorem below.

Lastly, Theorem 8 gives a bound on the computational complexity of on-line learning (in terms of m). In particular, the total computation time required to process the first m examples is only proportional to $m \log^c m$, for some constant c. Thus, in a sense, the "amortized" or "average" computation time on the m-th trial is only poly-logarithmic in m. (In fact, a more careful analysis would show that this is also true of the worst case computation time on the m-th trial.)

THEOREM 8. Let $\boldsymbol{C}$ be a learnable concept class. Then there exists an efficient on-line learning algorithm for $\boldsymbol{C}$ with the properties that:

- the probability of a mistake on the m-th trial is at most $m^{-1} \cdot p_1(n, s, \log m)$,
- the expected number of mistakes on the first m trials is at most $p_2(n, s, \log m)$,
- the total computation time required on the first m trials is at most $m \cdot p_3(n, s, \log m)$, and
- the space used on the first m trials is at most $p_4(n, s, \log m)$,

for some polynomials p_1, p_2, p_3, p_4.

Proof. Since $\boldsymbol{C}$ is learnable, there exists an efficient (batch) algorithm satisfying the properties of Theorem 4. Let A be such an algorithm, but with $\epsilon/2$ substituted for both ϵ and δ. Then the chance that A's output hypothesis incorrectly classifies a randomly chosen instance is at most ϵ. (This technique is also used by Haussler, Kearns, Littlestone and Warmuth (1988).)

Fix n and s, and let $m(\epsilon)$ be the number of examples needed by A. From Theorem 4, $m(\epsilon) \leq (p/\epsilon) \cdot \lg^c(1/\epsilon)$ for some constant c and some value p implicitly bounded by a polynomial in n and s. Let $\epsilon(m) = (p/m) \cdot \lg^c(m/p)$. Then it can be verified that $m(\epsilon(m)) \leq m$ for $m \geq 2p$. Thus, m examples suffice to find an hypothesis whose chance of error is at most $\epsilon(m)$.

To convert A into an on-line learning algorithm in a manner that preserves time and space efficiency, imagine breaking the sequence of trials into blocks of increasing size: the first block consists of the first $2p$ trials, and each new block has twice the size of the last. Thus, in general, the i-th block has size $s_i = 2^i p$, and consists of trials $a_i = 2(2^{i-1} - 1)p + 1$ through $b_i = 2(2^i - 1)p$.

On the trials of the i-th block, algorithm A is simulated to compute the i-th hypothesis h_i. Specifically, A is simulated with ϵ set to $\epsilon(s_i)$, which thus bounds the probability that h_i misclassifies a new instance. (Note that there are enough instances available in this block for A to compute an hypothesis of the desired accuracy.) On the next block, as the $(i + 1)$st hypothesis is being computed, h_i is used to make predictions; at the end of this block, h_i is discarded as h_{i+1} takes its place.

Thus, if the m-th trial occurs in the i-th block (i.e., if $a_i \leq m \leq b_i$), then the probability of a mistake is bounded by $\epsilon(s_{i-1})$, the error rate of h_{i-1}. From the definition of $\epsilon(\ )$, this implies the desired bound on the probability of a mistake on the m-th trial, and, in turn, on the expected number of mistakes on the first m trials.

Finally, note that on the i-th block, space is needed only to store the hypothesis from the last block h_{i-1}, and to simulate A's computation of block i's hypothesis. By Theorem 4, both of these quantities grow polynomially in $\log(1/\epsilon)$. By our choice of ϵ, this implies the desired bound on the algorithm's space efficiency. The time complexity of the procedure is bounded in a similar fashion. ■

7. Conclusions and open problems

We have shown that a model of learnability in which the learner is only required to perform slightly better than guessing is as strong as a model in which the learner's error can be made arbitrarily small. The proof of this result was based on the filtering of the distribution in a manner causing the weak learning algorithm to eventually learn nearly the entire distribution. We have also shown that this proof implies a set of general bounds on the complexity of PAC-learning (both batch and on-line), and have discussed some of the applications of these bounds.

It is hoped that these results will open the way on a new method of algorithm design for PAC-learning. As previously mentioned, the vast majority of currently known algorithms work by finding a hypothesis consistent with a large sample. An alternative approach suggested by the main result is to seek instead a hypothesis covering slightly more than half the distribution. Perhaps, such an hypothesis is easier to find, at least from the point of view of the algorithm designer. This approach leads to algorithms with a flavor similar to the one described for k-term DNF in Section 5.3, and it is possible to find similar algorithms for a number of other concept classes that are already known to be learnable (for example, k-decision lists (Rivest, 1987) and rank r decision trees (Ehrenfeucht and Haussler, 1989)). To what extent will this approach be fruitful for other classes not presently known to be learnable? This is an open question.

Another open problem concerns the robustness of the construction described in this paper. Intuitively, it seems that there should be a close relationship between reducing the error of the hypothesis, and overcoming noise in the data. Is this a valid intuition? Can our construction be modified to handle noise?

Finally, turning away from the theoretical side of machine learning, we can ask how well our construction would perform in practice. Often, a learning problem (for instance, a neural network) is designed, implemented, and found empirically to achieve a "good" error rate, but no way is seen of improving the program further to enable it to achieve a "great" error

rate. Suppose our construction is implemented on top of this learning program. Would it help? This is not a theoretical question, but one that can only be answered experimentally, and one that obviously depends on the domain and the underlying learning program. Nevertheless, it seems plausible that the construction might in some cases give good results in practice.

Acknowledgments

This paper was prepared with support from ARO Grant DAAL03-86-K-0171, DARPA Contract N00014-89-J-1988, and a grant from the Siemens Corporation.

Thanks to Sally Goldman, Michael Kearns, and Ron Rivest for their helpful comments and suggestions. Thanks also to the anonymous referees of this paper for their careful reading and thoughtful comments.

References

Angluin, D. (1980). Finding patterns common to a set of strings. *J. of Computer and System Sciences, 21*, 46–62.

Angluin D. (1988). Queries and concept learning. *Machine Learning, 2*, 319–342.

Angluin, D. and Valiant, L.G. (1979). Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Computer and System Sciences, 18*, 155–193.

Baum, E.B. (1989). On learning a union of half spaces. Unpublished manuscript.

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

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *J. of the Association for Computing Machinery, 36*, 929–965.

Board, R. and Pitt, L. (1990). On the necessity of Occam algorithms. (In press) *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*. New York, NY: ACM Press.

Boucheron, S. and Sallantin, J. (1988). Some remarks about space-complexity of learning, and circuit complexity of recognizing. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 125–138). San Mateo, CA: Morgan Kaufman.

Ehrenfeucht, A. and Haussler, D. (1989). Learning decision trees from random examples. *Information and Computation, 3*, 231–246.

Floyd, S. (1989). Space-bounded learning and the Vapnik-Chervonenkis dimension. *Proceedings of the Second Annual Workshop on Computational Learning Theory* (pp. 349–364). San Mateo, CA: Morgan Kaufman.

Haussler, D. (1988). *Space efficient learning algorithms* (Technical Report UCSC-CRL-88-2). Santa Cruz, CA: University of California, Baskin Center for Computer Engineering and Information Sciences.

Haussler, D., Kearns, M., Littlestone, N., and Warmuth, M.K. (1988). Equivalence of models for polynomial learnability. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 42–55). San Mateo, CA: Morgan Kaufman.

Haussler, D., Littlestone, N., and Warmuth, M.K. (1987). Expected mistake bounds for on-line learning algorithms. Unpublished manuscript.

Haussler, D., Littlestone, N., and Warmuth, M.K. (1988). Predicting {0, 1}-functions on randomly drawn points. *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science* (pp. 100–109). Washington, DC: IEEE Computer Society Press.

Helmbold, D., Sloan, R., and Warmuth, M.K. (1990). Learning nested differences of intersection-closed concept classes. *Machine Learning, 5*, xxx–xxx.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *J. of the American Statistical Association, 58*, 13–30.

Kearns, M. (1988). Thoughts on hypothesis boosting. Unpublished manuscript.

Kearns, M. (1989). *The Computational Complexity of Machine Learning*. Doctoral dissertation, Department of Computer Science, Harvard University, Cambridge, MA.

Kearns, M., Li, M., Pitt, L., and Valiant, L. (1987). On the learnability of Boolean formulae. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 285–295). New York, NY: ACM Press.

Kearns, M. and Valiant, L.G. (1988). *Learning Boolean formulae or finite automata is as hard as factoring* (Technical Report TR-14-88). Cambridge, MA: Harvard University Aiken Computation Laboratory.

Kearns, M. and Valiant, L.G. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (pp. 433–444). New York, NY: ACM Press.

Pitt, L. and Valiant, L.G. (1988). Computational limitations on learning from examples. *J. of the Association for Computing Machinery, 35*, 965–984.

Rivest, R.L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Schapire, R.E. (1989). Pattern languages are not learnable. Unpublished manuscript.

Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142.