Annu. Rev. Comput. Sci. 1990. 4:255–306

MACHINE LEARNING

Thomas G. Dietterich

Department of Computer Science, Oregon State University, Corvallis, Oregon 97331-3902

CONTENTS

OVERVIEW

Recent progress in the study of machine learning methods has taken many directions. First, in the area of inductive learning, a new formal definition of learning introduced by Leslie Valiant has provided the foundation for several important theoretical results. Second, a number of new learning algorithms have been developed, and existing algorithms have been improved. Third, the collection of methods that perform so-called explanation-based learning have addressed the problem of speeding up the per-

8756–7016/90/1115-0255$02.00

formance of problem-solving programs. Finally, the philosophical foundations of machine learning have been clarified.

My goal here is to review the major results in each of these four directions. We begin with a discussion of the field's philosophical foundations, which provide a framework for the remainder of the chapter.

PHILOSOPHICAL FOUNDATIONS

How can "learning" be defined? The lack of a workable definition has made it hard to determine whether learning methods succeed. Recently, Dietterich (1986) and Valiant (1984) have introduced new approaches to defining "learning."

Dietterich (1986) reduces the problem of defining "learning" to the problem of defining "knowledge." Given a satisfactory definition of "knowledge," "learning" can be defined as an increase in "knowledge." Depending on which definition of "knowledge" one chooses, one obtains different definitions of learning.

The definition I prefer is the following. An agent (i.e. a person or a program) knows a fact F if the agent has been told F or if the agent can logically infer F from its other knowledge. No limit is placed on the computational resources (e.g. CPU time and memory space) consumed in performing these inferences. This form of knowledge can be called "knowledge in principle" or "deductive closure knowledge." The logical inferences are assumed to preserve correctness (i.e. they are monotonic, deductive inferences).

Given this definition of knowledge, learning (i.e. increases in knowledge) can occur under two circumstances. Learning occurs when the agent is told a fact F that it did not know and when the agent makes an "inductive leap" and chooses to believe some fact F that is not entailed by its existing knowledge.

For example, suppose an agent knows that a poker hand containing three Jacks is superior to a hand containing only two Queens. Suppose the agent also knows that a poker hand containing three Tens is superior to a hand containing two Eights. Learning occurs when the agent jumps to the conclusion that any hand containing three cards of rank R_1 is superior to any hand containing at most two cards of rank R_2. In short, a system that formulates general rules by analyzing specific examples is one kind of learning system.

Notice that according to this definition, learning does not take place if a system discovers a more efficient way to infer a fact that it already knows in principle. Consequently, simple speed-ups (e.g. such as those obtained by caching inferences) do not count as learning. However, this definition

also has the unfortunate consequence that a program that knows the rules of chess would also know the optimal strategy.

Another definition of knowledge involves the notion of "explicit belief" suggested by Fagin & Halpern (1987). According to this definition, an agent has a combination of implicit beliefs (these correspond to the deductive closure definition of "knowledge" discussed above) and explicit beliefs (i.e. beliefs the system is "aware" of). Logical (monotonic) inference can make implicit beliefs explicit. In a particular program, one might define a belief as explicit if it is stored in a database or if it can be computed within a fixed time limit. Learning takes place, according to this definition, whenever new explicit beliefs are found. Hence, this definition does include simple speed-ups (e.g. those produced by traditional programming language compilers) as forms of learning. It does not draw a distinction between learning as efficiency improvement and learning as the acquisition of a new rule from examples.

By considering these definitions of "knowledge" and "learning," we can develop a three-part taxonomy of learning systems: (*a*) systems that receive no inputs and simply become more efficient over time (*speed-up learning*), (*b*) systems that receive new knowledge via inputs but otherwise perform no inductive leaps (*learning by being told*), and (*c*) systems that perform inductive leaps to acquire knowledge that was not previously known either explicitly or implicitly (*inductive learning*).

These definitions provide a basis for evaluating learning systems. Speed-up learning systems should be evaluated by measuring the efficiency improvement they produce. Systems that learn by being told can be evaluated according to their ability to exploit the information they receive. Finally, inductive learning systems must be evaluated according to the correctness of the knowledge they produce. This is difficult, however, because inductive learning systems can provide no guarantee of correctness unless they cease to make inductive leaps!

Leslie Valiant's probabilistic framework (Valiant 1984) provides a solution to this last difficulty. Valiant says that a system has learned a fact F if it can guarantee with high probability that F is approximately correct. This definition relaxes the goal of guaranteed correctness in two ways. First, the fact F is permitted to be only approximately correct. Second, with low probability, the learning system may produce an hypothesis F that is totally incorrect. It turns out that this definition provides us with a rigorous criterion for evaluating learning programs.

To understand what it means to be "approximately correct," let us view a fact F as a relation over some universe U of objects. In other words, F is the subset of objects (or tuples) in U that make F true. Intuitively, a second fact $\hat{F}$ is approximately correct if the symmetric difference $F \oplus \hat{F}$

is small (this corresponds to the shaded region in Figure 1). In other words, F and $\hat{F}$ agree over most of the universe U.

Valiant elaborates this definition by taking into consideration the possibility that some elements of U are more important than others. He considers $\hat{F}$ to be approximately correct to the degree that it matches F on the more important elements of U. Specifically, Valiant assumes that the learning system is going to be confronted with a series of "performance trials." In each trial, it will be presented an element $u \in U$ and asked whether $u \in F$ is true. Let P be an unchanging probability distribution over U such that $P(u)$ is the probability that u will be selected in any given trial. Then error$(F, \hat{F})$ is defined to be the probability that the learning system will make a mistake in any given performance trial. Formally,

$$\text{error}(F, \hat{F}) = \sum_{u \in F \oplus \hat{F}} P(u).$$

The fact $\hat{F}$ is approximately correct if error$(F, \hat{F})$ is less than ε, where ε is a small constant called the *accuracy parameter.*

Now that we understand what it means to be "approximately correct," we must consider the second part of Valiant's definition: The learning system that produces $\hat{F}$ may itself make mistakes from time to time and produce hypotheses that are not approximately correct. In particular, the learning system is usually constructing $\hat{F}$ by analyzing a collection of training examples. A training example is a pair of the form (u, c), where $u \in U$ and $c = 1$ if $u \in F$ and $c = 0$ otherwise. If those examples do not provide a representative sample of F, then the learning program may come up with a bad guess, $\hat{F}$.

By making some assumptions about the training sample, we can bound the probability that the learning system will produce an $\hat{F}$ with error greater

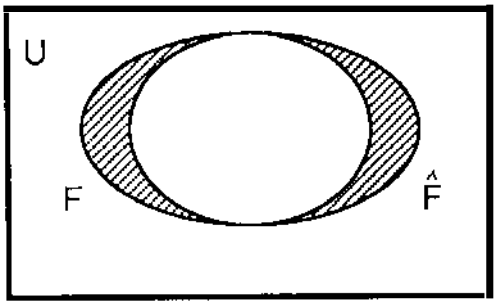

Figure 1 The error between the correct fact F and $\hat{F}$.

than ε. Specifically, let us assume that the training sample is constructed by independently drawing m examples from U according to the same probability distribution $P(u)$ that will be used during the performance trials. We say that the learning system is *probably approximately correct* (*PAC*) if

$$Pr[\text{error}(F, \hat{F}) > \varepsilon] < \delta,$$

where δ is called the *confidence parameter* and where the probability is taken over all training samples of size m.

What Valiant has done is to incorporate a notion of evidential support into the definition of "learning." According to Valiant, a program is not considered a learning program if it makes a lucky leap and comes up with a correct fact. Instead, Valiant requires that the learning program consider a large enough set of training examples so that its hypothesis $\hat{F}$ is statistically justified.

This is a major breakthrough because it provides a standard against which to compare inductive learning programs. It also provides a basis for proving results concerning the computational tractability of various learning problems. These results are the topic of the next section.

THEORETICAL RESULTS ON LEARNING FROM EXAMPLES

As we have seen above, the goal of learning from examples is to infer, from a set S of training examples, a probably approximately correct fact $\hat{F}$.[1] In principle, this is impossible, because the knowledge of whether $F(u)$ is true for one point in U tells us nothing about the values of F at any other points in U—it merely tells us the value of F at u. When people are confronted with such problems, they circumvent them by imposing some assumptions concerning F. They may assume, for example, that F can be represented as a Boolean conjunction over the features describing U. Or they may prefer the simplest hypothesis $\hat{F}$ consistent with the training examples. This amounts to assuming that F can be represented simply in some given language.

These assumptions concerning F are called the "bias" of the learning system, and they provide it with some means for making a guess concerning the identity of F. There are two general forms of bias: restricted-hypothesis-space bias and preference bias.

[1] This terminology is informal. Technically, we should say that $\hat{F}$ is produced by an *algorithm* that is probably approximately correct.

Under the restricted-hypothesis-space bias, the learning system assumes that the correct concept F is a member of some hypothesis set H, where H contains only some of the $2^{|U|}$ possible concepts over U. This is usually implemented by assuming that F has some restricted syntactic form (e.g. is a Boolean conjunction).

Under the preference bias, the learner imposes a preference ordering over the set of hypotheses and attempts to find the "best" hypothesis $\hat{F}$ according to this ordering. In this chapter we will assume that the preference ordering is a total ordering, and we will let index $I(\hat{F})$ denote the numerical position of $\hat{F}$ in this ordering. The preference bias can be implemented by attempting to find a consistent hypothesis $\hat{F}$ of low index.

Restricted-Hypothesis-Space Bias

Suppose we are given m training examples labeled according to the correct concept F. The examples are drawn independently from U according to some unknown probability distribution $P(u)$. We are also given a restricted hypothesis space H. Our algorithm will attempt to find an hypothesis $\hat{F} \in H$ that is consistent with all m training examples. Assuming that such an $\hat{F}$ can be found, what is the probability that it has error greater than ε?

To answer this question, let us define the set $H_{bad} = \{h_1, \ldots, h_l\}$ to be the set of hypotheses in H that have error greater than ε. We will compute the probability that, after m examples have been processed, there is some element of H_{bad} that is consistent with the training examples. If this probability is small enough, then (with high probability) the only hypotheses remaining in H that are consistent with the training examples are hypotheses with error less than ε. Hence, if our learning algorithm finds a consistent hypothesis $\hat{F} \in H$, that hypothesis is probably approximately correct.

Let us begin by considering a particular element $h_1 \in H_{bad}$. What is the probability that h_1 is consistent with one randomly drawn training example? It is just the probability that the training example was drawn from the region of U *outside* the shaded area of Figure 1. This probability is greatest when $\text{error}(F, h_1) = \varepsilon$. That is, h_1 is as good as possible without being approximately correct. So, the probability that h_1 is consistent with a single training example is no more than $1 - \varepsilon$.

It follows that the probability that h_1 is consistent with all m randomly drawn training examples is no more than $(1-\varepsilon)^m$. We will write this as $P^m[\text{consist}(h_1)] \leq (1-\varepsilon)^m$.

Now let us consider all of the hypotheses in H_{bad}. What is the probability that after m examples there is some element of H_{bad} that has not been eliminated from consideration? This is

$$P^m[\text{consist}(H_{bad})] = P^m[\text{consist}(h_1) \vee \cdots \vee \text{consist}(h_1)].$$

Because the probability of a disjunction (union) of several events is no larger than the sum of the probabilities of each individual event,

$$P^m[\text{consist}(H_{bad})] \leq |H_{bad}| \cdot (1-\varepsilon)^m.$$

In the worst case, $H_{bad} = H$ (i.e. there are no approximately correct hypotheses in H). Hence,

$$P^m[\text{consist}(H_{bad})] \leq |H|(1-\varepsilon)^m.$$

Now that we have an expression for the probability that $\hat{F}$ is not approximately correct, we can set this equal to δ and solve for m to obtain a bound on the number of training examples to guarantee that $\hat{F}$ is probably approximately correct.

$$|H|(1-\varepsilon)^m \leq \delta$$

is true if and only if

$$m \geq \frac{1}{-\ln(1-\varepsilon)}\left(\ln\frac{1}{\delta} + \ln|H|\right).$$

But since $\varepsilon \leq -\ln(1-\varepsilon)$ over the interval $[0, 1)$, it suffices that

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right).$$

This gives us Theorem 1:

Theorem 1 (Blumer et al 1987) *Let H be a set of hypotheses over a universe U, S be a set of m training examples drawn independently according to $P(u)$, ε, $\delta > 0$, then if $\hat{F} \in H$ is consistent with all training examples in S and*

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$$

then the probability that $\hat{F}$ has error greater than ε is less than δ.

Using this theorem, we can obtain bounds on the number of examples required for learning in various hypothesis spaces. Consider, for example, the set of hypotheses H_{conj} that can be expressed as simple conjunctions of n Boolean variables. There are 3^n such hypotheses, since in a conjunction each variable may appear negated, un-negated, or it may be missing. Applying Theorem 1, we see that if

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + n\ln 3\right)$$

then any hypothesis consistent with the examples will be PAC. Furthermore, the number of examples required grows only linearly with the number of features.

Likewise, consider the set of hypotheses that can be expressed as linear threshold functions over n Boolean variables, $x_1, \ldots, x_n$. A linear threshold function is described by a vector of real-valued weights, $w_1, \ldots, w_n$ and a real-valued threshold, θ. It returns a 1 if $\sum_{i=1}^{n} w_i x_i \geq \theta$. Muroga (1971), shows that $|H| \leq 2^{n^2}$. Hence, if

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + n^2\ln 2\right)$$

then any linear threshold function consistent with the training examples is PAC.

Table 1 shows the hypothesis space sizes for several popular concept representations. The class k-term-DNF contains Boolean formulas in disjunctive normal form with at most k disjuncts (i.e. a k-term disjunction where each term is a conjunction of unlimited size). The class k-DNF contains Boolean formulas in disjunctive normal form in which each conjunction has at most k variables (i.e. a disjunction of any number of conjunctive terms, but each conjunction is limited to length k). A class analogous to k-DNF is the class k-CNF. Each formula in k-CNF is a conjunction of clauses (disjunctions). Each clause contains at most k variables. The class k-DL is the class of decision lists introduced by Rivest (1987). A decision list is an ordered list of pairs of the form $\langle (F_1, C_1), \ldots, (F_i, C_i), \ldots, (T, C_{l+1})\rangle$. Each F_i is a Boolean conjunction of at most k

Table 1 Sizes of various concept description languages

Hypothesis space	Size
Boolean conjunctions	3^n
k-term-DNF	$2^{O(kn)}$
k-DNF	$2^{O(n^k)}$
k-CNF	$2^{O(n^k)}$
k-DL	$2^{O(n^k k \lg n)}$
LTU	$2^{O(n^2)}$
DNF	2^{2^n}

variables, and each C_i indicates the result (either 0 or 1). A decision list is processed like a Lisp COND clause. The pairs are considered in order until one of the F_i is true. Then the corresponding C_i is returned as the result. By convention, the condition for the last pair in the list, F_{l+1}, is always true (T). The class LTU contains all Boolean functions that can be represented by linear threshold units.

For comparison, we also show the full class DNF, consisting of any arbitrary Boolean expression in disjunctive normal form. DNF is capable of representing any of the Boolean functions.

Note that for fixed k, each of these classes (except DNF) requires only a polynomial number of training examples to guarantee PAC learning according to Theorem 1.

Theorem 1 gives results for finite hypothesis spaces. However, there are many applications in which hypotheses contain real-valued parameters, and consequently there are uncountably many hypotheses in these spaces. In spite of this, it is still possible to develop learning algorithms for these cases. Consider for example the universe U consisting of points on the real number line. An hypothesis $F \subset U$ describes some subset of these points. Suppose we restrict our hypotheses to be single closed intervals over the real line (i.e. our hypotheses have the form $[a, b]$). One algorithm for discovering closed intervals would be to let a be the value of the smallest positive example and b be the value of the largest positive example. How many training examples are needed to ensure that this algorithm will return an interval that is probably approximately correct?

Answers for problems such as this can be obtained using a measure of bias called the Vapnik-Chervonenkis dimension (VC-dimension). The idea behind the VC-dimension is that although an hypothesis space may contain uncountably many hypotheses, those hypotheses may still have restricted expressive power. Specifically, we will say that a set of hypotheses can *completely fit* a collection of examples $E \subset U$ if, for every possible way of labeling the elements of E positive or negative, there exists an hypothesis in H that will produce that labeling. The VC-dimension will be defined to be the size $|E|$ of the largest set of examples that H can completely fit. This will provide a measure of the expressive power of H.

To continue with the real-interval illustration, let us consider the set of two points $E = \{3, 4\}$. There are four different ways that these two points can be labeled as positive or negative, corresponding to four different training sets:

$$S_0 = \{\langle 3, 0\rangle, \langle 4, 0\rangle\}$$

$$S_1 = \{\langle 3, 0\rangle, \langle 4, 1\rangle\}$$

$S_2 = \{\langle 3,1\rangle, \langle 4,0\rangle\}$

$S_3 = \{\langle 3,1\rangle, \langle 4,1\rangle\}$.

For each possible labeling, there is a real interval that will produce that labeling:

S_0 can be labeled by [0, 1]

S_1 can be labeled by [4, 5]

S_2 can be labeled by [2, 3]

S_3 can be labeled by [2, 5].

Hence, the hypothesis space H_{int} consisting of closed intervals on the real line can *completely fit* the set E. Indeed, it is easy to see that any set of two points can be fitted completely by H_{int}.

However, consider the set of points $E' = \{2, 3, 4\}$. The hypothesis space H_{int} cannot completely fit this set. In particular, there is no hypothesis in H_{int} that can label E' as follows:

$S_4 = \{\langle 2,1\rangle, \langle 3,0\rangle, \langle 4,1\rangle\}$.

This is because any interval containing 2 and 4 will also contain 3.

Since the VC-dimension of H is defined as the largest set of points that H can completely fit, it is easy to see that $\text{VCdim}(H_{int}) = 2$.

A more interesting example concerns linear threshold units over arbitrary points in n-dimensional Euclidean space. A linear threshold unit is equivalent to a hyperplane that splits R^n into two half-spaces. If a given set of training examples can be separated such that the positive examples are all on one side of the hyperplane and the negative examples are all on the other side, then the training examples are said to be *linearly separable*. When $n = 2$, it is easy to see that half-spaces (in this case, half-planes) can completely fit any set of three points. However, half-planes are unable to completely fit any collection of four points (i.e. some labelings of the points will not be linearly separable). In general, the VC-dimension for linear threshold units over n-dimensional Euclidean space is $n+1$.

Intuitively, the VC-dimension is proportional to the logarithm of the size of the *effective* hypothesis space. Indeed, the following theorem shows how Theorem 1 can be extended using the VC-dimension:

Theorem 2 (Blumer et al 1989) *A set of hypotheses H is PAC learnable if*

$$m \geq \frac{1}{\varepsilon}\max\left[4\lg\frac{2}{\delta}, 8 \cdot VCdim(H)\lg\frac{13}{\varepsilon}\right]$$

and the algorithm outputs any hypothesis $\hat{h} \in H$ *consistent with* S.

Using Theorem 2, we can tighten the bound on the number of examples required for learning linear threshold units to $O(1/\varepsilon(n \ln 1/\varepsilon + 1/\delta))$.

Perhaps the most interesting application of Theorem 2 (and its relatives) is to the problem of training feed-forward multi-layer neural networks. A difficulty with the practical application of these networks is to decide how large the network should be for each application. If the network is too large, it is easy to find a setting of the weights that is consistent with the training examples. However, the resulting network is unlikely to classify additional points in U correctly.

Baum & Haussler (1988) consider feed-forward networks of N linear threshold units and W weights. They show that if the weights can be set so that at least a fraction $1-(\varepsilon/2)$ of the m training examples are classified correctly and if

$$m \geq O\left(\frac{W}{\varepsilon}\log\frac{N}{\varepsilon}\right),$$

then the network is PAC with $0 < \varepsilon < 1$ and $0 < \delta < O(e^{-\varepsilon m})$.

The VC-dimension turns out to be a fundamental notion. It permits us to exactly characterize the set of learnable concepts, and it allows us to derive a lower bound on the number of examples needed for learning. These results are given in the following two theorems.

Theorem 3 (Blumer et al 1989) *A space of hypotheses* H *is PAC learnable iff it has finite Vapnik-Chervonenkis (VC) dimension.*[2]

Theorem 4 (Ehrenfeucht et al 1988) *Any PAC learning algorithm for* H *must examine*

$$\Omega\left(\frac{1}{\varepsilon}\left[\ln\frac{1}{\delta} + VCdim(H)\right]\right)$$

training examples.

[2] It is possible to learn concept classes having infinite VC-dimension if the number of training examples is permitted to vary with the complexity of the concepts in the hypothesis space. See Linial et al (1989).

Preference Bias

With Theorems 1–4, we have a fairly complete understanding of learning with a restricted-hypothesis-space bias. Let us now briefly turn our attention to the problem of learning with a preference bias. Recall that a preference bias establishes an ordering over all of the hypotheses in H. We will let the index $I(F)$ be the numerical position of hypothesis F in this ordering. By definition, hypotheses with smaller index values $I(F)$ will be considered simpler than hypotheses with higher index values.

Now suppose we have an excellent learning algorithm that works as follows. For any given set of training examples S, it finds the hypothesis $\hat{F} \in H$ of lowest index that is consistent with S. It turns out that if the number of examples in S is sufficiently large and if the hypothesis found by the algorithm has sufficiently small index, then we can be quite confident that $\hat{F}$ is approximately correct. The reason is that for sufficiently large S, it is unlikely that we could have found such a simple (i.e. small index) hypothesis $\hat{F}$ that is consistent with the training examples.

Following (Blumer et al 1987), we can formalize this by letting H' be the space of hypotheses of index less than or equal to $I(\hat{F})$. The set H' can be viewed as the effective hypothesis space for our preference-bias algorithm for this particular sample S, and therefore, from Theorem 1, we can conclude that the number of examples required is

$$\frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln I(\hat{F})\right).$$

This result can be generalized to allow the learning algorithm to output an hypothesis $\hat{F}$ that has small, but not minimal, index. See Blumer et al (1987) for details.

The famous bias of Occam's Razor (prefer the simplest hypothesis consistent with the data) can thus be seen to have a mathematical basis. If we choose our simplicity ordering *before* examining the data, then a simple hypothesis that is consistent with the data is provably likely to be approximately correct. This is true regardless of the nature of the simplicity ordering, because no matter what the ordering, there are relatively few simple hypotheses. Therefore, a simple hypothesis is unlikely to be consistent with the data by chance.

Another way of thinking about this result is to view learning programs as data compression algorithms. They compress the training examples into an hypothesis, $\hat{F}$, by taking advantage of some predefined encoding scheme (i.e. simplicity ordering). If the data compression is substantial (i.e. the number of bits needed to represent the hypothesis is much smaller than

the number of training examples), then the hypothesis is likely to be approximately correct.

Noisy Data

All of the results described above have assumed that the training examples are complete and correct. Unfortunately, there are many applications where the training data are incomplete and incorrect. For incorrect training examples—that is, examples that are incorrectly classified—all of the results discussed above can be generalized as follows. Instead of trying to find a concept $\hat{F} \in H$ that is consistent with all of the training examples, it suffices to find an $\hat{F}$ that is consistent with fraction $1-(\varepsilon/2)$ of the training examples. Theorems 1–4 still apply under these conditions with some slight adjustments (see Appendix 3 of Blumer et al 1989).

Computational Complexity

So far we have considered only what is called *sampling complexity*—that is, the number of training examples required to guarantee PAC learning. A second aspect of learning has also been investigated within the Valiant framework: the *computational complexity* of finding an hypothesis in H consistent with the training examples.

If we look again at Theorem 1, we see that the number of examples required for learning is proportional to the log of the size of the hypothesis space. This means that with a linear number of examples, we can learn an exponential number of hypotheses. The most trivial algorithm for finding an hypothesis consistent with the examples would simply enumerate each hypothesis in H and test it for consistency with the examples. However, when there are exponentially many hypotheses, this approach will require exponential time. Therefore, the challenge is to find ways of computing a consistent hypothesis by analyzing the training examples more directly. Our goal is to find algorithms that require time polynomial in the number of input features n and in $1/\varepsilon$ and $1/\delta$.

Table 2 shows the computational complexities for the best known algorithms for several hypothesis spaces. Following Valiant, we say that an hypothesis space H is polynomially learnable if (*a*) only a polynomial number of training examples are required (as a function of n, $1/\varepsilon$, and $1/\delta$) and (*b*) a consistent hypothesis from H can be found in time polynomial in n, $1/\varepsilon$, and $1/\delta$. Hence, from the table, we can see that conjunctions, k-DNF, k-DL, and the linear threshold units are all polynomially learnable. The hypothesis space k-3NN consists of feed-forward neural networks containing two layers of linear threshold units (often called three-layer networks). The first layer of units (usually called the "hidden layer") contains exactly k units. There are robust proofs that this hypothesis space

Table 2 Computational complexity of finding a consistent hypothesis

Hypothesis space	Time complexity
Boolean conjunction	Polynomial
k-term-DNF	NP-hard
k-DNF	Polynomial
k-CNF	Polynomial
k-DL	Polynomial
LTU	Polynomial
k-3NN	NP-hard

is not polynomially learnable (Judd 1987, 1988; Blum & Rivest 1988; Lin & Vitter 1989).

As an example of a polynomial-time learning algorithm, consider the following algorithm for learning Boolean conjunctions. We will represent a conjunction C as a list of Boolean variables or their negations. Given a collection S of training examples, we find the first positive example p_1 in that list and initialize C to contain all of the variables (or their negations) present in that positive example (if there are no positive examples, we exit and guess the null concept, $x_1 \wedge \neg x_1$). Then for each additional positive example p_i, we delete from C any Boolean variables appearing in p_i with a sign different from their sign in C. After processing all of the positive examples, we check all of the negative examples to make sure that none of them are covered by C. Finally, we return C as the answer.

Consider the following positive examples:

$\langle(0\ 1\ 1\ 0), 1\rangle$

$\langle(1\ 1\ 1\ 0), 1\rangle$

$\langle(1\ 1\ 0\ 0), 1\rangle$.

After processing the first example, $C = \{\neg x_1, x_2, x_3, \neg x_4\}$; after processing the second example, $C = \{x_2, x_3, \neg x_4\}$; after the third example, $C = \{x_2, \neg x_4\}$. This algorithm requires $O(nm)$ steps.

Surprisingly, smaller hypothesis spaces are not always easier to learn. For example, the space k-term-DNF is a proper subspace of the k-CNF, yet k-CNF is polynomially learnable but k-term-DNF is not (Pitt & Valiant 1988). Similarly, the space of Boolean threshold units (i.e. linear threshold units in which the weights are all Boolean) is not polynomially learnable, but LTU (which properly contains it) is. One explanation for this is that, in some cases, by enlarging the hypothesis space it becomes easier to find an hypothesis consistent with the training examples. The larger space

provides more freedom to choose the syntactic form of the hypothesis. Another explanation is that different representations, even of the same space, have different computational properties. Hence, some representations for concepts are easier to relate to the representation of the training examples.

These observations indicate that if we want to prove that learning a concept class is computationally intractable, we need to show that it is intractable *regardless of the representation employed by the learning algorithm.* In other words, suppose the correct concept F can be represented by a k-term-DNF formula. Although the problem of finding a k-term-DNF formula consistent with a training sample for F is NP-complete, we know that in polynomial time we can find an $\hat{F}$ represented as an equivalent k-CNF formula. Hence, we can construct an algorithm that can learn every concept in k-term-DNF by using hypotheses represented in k-CNF.

This point is particularly important for classes, such as k-3NN, where although it is intractable to find a consistent hypothesis using k hidden units, it might be easier to find a consistent hypothesis using $k' > k$ hidden units. If k' is only moderately bigger than k, the number of training examples required to guarantee PAC learning would still be polynomial. In general, if s is the number of bits required to represent the correct hypothesis F, then any algorithm that can represent $\hat{F}$ using $p(s)$ bits (where p is some polynomial) will still have polynomial sample complexity.

The question of whether every concept in k-3NN can be learned by finding (in polynomial time) a concept in k'-3NN (where $k' \leq p(k)$ for some polynomial p) is open. However, for two other important concept classes, the analogous questions have been answered negatively.

Let $DFA(s)$ be the space of concepts that can be represented as deterministic finite state automata of size $\leq s$. If S is a training sample for a concept $F \in DFA(s)$, then the problem of finding an hypothesis $\hat{F} \in DFA(p(s))$ consistent with S, for some polynomial p is NP-complete (Pitt & Warmuth 1989).

Similarly, if $BF(s)$ is the space of concepts that can be represented as Boolean formulas of size $\leq s$ and if S is a training sample for a concept $F \in BF(s)$, then the problem of finding an hypothesis $\hat{F} \in BF(p(s))$ consistent with S, for some polynomial p, is as hard as factoring integers (Kearns & Valiant 1988, 1989). In fact, this result can be strengthened to apply to *any* representation language in which $\hat{F}$ has size $\leq p(s)$.

An important way of looking at these results is from the perspective of Occam's Razor. Consider the class of all Boolean formulas and suppose we adopt the bias of preferring shorter formulas. The problem of finding the smallest Boolean formula consistent with a set of training examples has long been known to be NP-complete (Gold 1978). However, we might

settle for an approximation to Occam's Razor—we could accept any Boolean formula that is of size $\leq p(s)$, where s is the size of the smallest Boolean formula consistent with the data. If we assume that factoring is hard, these results imply that there is no polynomial time algorithm for finding these "nearly simplest" hypotheses.

In short, it appears that there are "simple" concepts (i.e. that can be represented by polynomial-sized finite state machines or regular expressions) that cannot be discovered by any learning algorithm using any representation. Nature may be simple, but (in the worst case) no computing device can reveal that simplicity in polynomial time (unless $P = NP$, of course).

Summary

The Valiant theory allows us to quantify the role of bias in inductive learning. The main implication of this theory is that there are no efficient, general-purpose inductive learning methods. Specifically, in order to learn using a polynomial number of training examples, by Theorem 4 the VC-dimension must be a polynomial function of n, $1/\varepsilon$, and $1/\delta$. The VC-dimension of the entire space of 2^{2^n} Boolean functions over n variables is clearly 2^n, so it is impossible to learn arbitrary Boolean functions using only a polynomial number of examples.

On the positive side, the theory states conditions under which we can determine, with high confidence, whether a given learning algorithm has succeeded. For a given bias, the theory says that *if* a consistent hypothesis $\hat{F} \in H$ can be found and the number of examples m is large enough, then $\hat{F}$ is probably approximately correct. Unfortunately, the hypothesis space H must constitute only a small fraction of the possible hypotheses, and therefore any particular learning algorithm is unlikely to succeed for a randomly chosen concept $F \subset U$. Indeed, it is because H is a small fraction of the space of possible hypotheses (2^U) that we can have statistical confidence in the results of the learning algorithm.

Hence, for a particular application, the vocabulary of features chosen to represent training examples and hypotheses must allow a consistent $\hat{F}$ to be found. In many applications (Michalski & Chilausky 1980; Quinlan et al 1986), this has turned out to be achieved easily, but there are others where it has been quite difficult (Quinlan 1983).

RECENT DEVELOPMENTS IN PRACTICAL LEARNING ALGORITHMS

Here we focus on three significant developments in practical learning algorithms: (*a*) improvements to decision tree–induction algorithms, (*b*)