Machine Learning 4, 67–97, 1989

On Learning Sets and Functions

B. K. NATARAJAN (NAT@CS.CMU.EDU)
The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.

Editor: Pat Langley

Keywords: Learning sets, learning functions, probabilistic analysis, connectionist networks.

Abstract. This paper presents some results on the probabilistic analysis of learning, illustrating the applicability of these results to settings such as connectionist networks. In particular, it concerns the learning of sets and functions from examples and background information. After a formal statement of the problem, some theorems are provided identifying the conditions necessary and sufficient for efficient learning, with respect to measures of information complexity and computational complexity. Intuitive interpretations of the definitions and theorems are provided.

1. Introduction

This paper concerns algorithms that learn sets and functions from examples. The results presented in these papers appeared in preliminary form in Natarajan (1986, 1987, 1988b) and in Natarajan and Tadepalli (1988). Among others, the following authors have reported related investigations: Angluin (1986), Rivest and Schapire (1987), Berman and Roos (1987), Laird (1987), Blumer, Ehrenfeucht, Haussler, and Warmuth (1986), and Kearns, Li, Pitt, and Valiant (1987a). Although there is some overlap of results between this paper and some of the aforementioned papers, the results in this paper represent independent developments that often favor simpler proof techniques.

Over the years, many papers in the literature have addressed the topic of concept learning. These papers can be broadly classified into two categories: (1) the formal work on inductive inference and, (2) the more empirical work in artificial intelligence. (For an excellent review of the inductive inference literature, see Angluin and Smith, 1983.) As it happened, the wide gap between the basic assumptions of inductive inference on the one hand and the needs of the empiricists on the other denied the formal work significant practical import. The most generous contribution to this gap may have come from an emphasis on worst-case[1] analysis by the inductive inference group.

More recently, Valiant (1984) introduced a formal framework for concept learning with a view towards probabilistic analysis. The framework is probabilistic in that it only requires the learning agent to learn with high probability, and having learned, to be correct with high probability. Furthermore, the teacher and the examiner are the same entity, doing away with the worst-case hopelessness of learning from one teacher and then having to face an unknown examiner with

leanings far divergent from the teacher. In some sense, this framework captures the essence of the concept-learning problem faced by humans and hence is of interest from both the theoretical and the practical viewpoint. The results in this paper are based on Valiant's framework and its variants. In the main, we will concentrate on the definitions and the intuition behind our theorems rather than on the proofs and technicalities involved, in the belief that the former will hold greater interest for the readers of this journal. However, in the interest of completeness, formal proofs are presented in appendices when warranted. Other authors have presented formal results of a similar flavor in the AI literature, including Rivest (1987), Angluin and Laird (1988), and Littlestone (1988).

We define a concept to be a subset of a universe of objects. (We will formalize this later.) An example for a concept is an object from the universe together with a label indicating whether or not the object belongs to the concept. If the object belongs to the concept, the example is positive; otherwise, it is negative. The primary aim of our study can be summarized as follows: suppose the learner were required to learn an unknown concept from examples. If the learner knew nothing *a priori* about the concept to be learned, then he can say nothing about objects that he has not seen as examples. In other words, unless the concept is exhaustively enumerated, the learner has little hope of learning the concept. On the other hand, if the learner knows the unknown concept is, say, one of two predetermined concepts, then a single well-chosen example would suffice. Thus the number of examples the learner requires to learn a concept is intimately linked with what he knows about the concept at the start of the learning process. Our question can be stated thus: What is the quantitative relationship between the number of examples required and the learner's *a priori* knowledge about a concept? We will attempt to answer this question, although we must first expend much effort to make it precise.

In Section 2 we give a formal definition of the learning framework. Specifically, the framework concerns learning concepts defined on the strings of a finite alphabet. In Section 3, we consider a simple and intuitive notion of the dimension of a space of concepts. We then use this notion of dimensionality to give theorems that quantitatively link the efficient learning of a space of concepts with the dimension of the space. Section 4 concerns the time complexity of learning, characterizing the class of spaces that can be learned efficiently. In Section 5, we discuss learning functions as opposed to learning concepts or sets, and show that our development of concept learning is smoothly extensible to this setting.

In Section 6, we modify our learning framework to permit the learner both examples for the target concept and hints on it. For instance, when teaching a concept in geometry, the teacher may present the learner with some basic theorems in geometry, in addition to some examples for the concept. Our main result here is a theorem that establishes the equivalence between this framework and the earlier one; i.e., with regard to information complexity, learning from examples and advice is no more powerful than learning from examples alone. This equivalence does not hold when we consider time complexity, as we can easily show that advice can

reduce the time complexity of learning substantially.

The results of Sections 2 through 6 concern functions and concepts on discrete spaces—strings on the Boolean alphabet. In contrast, Section 7 concerns sets and functions on continuous spaces. Blumer et al. (1986) use the notion of the Vapnik-Chervonenkis dimension for sets on continuous spaces to obtain a learnability theorem for concepts. We briefly review their results and then generalize the Vapnik-Chervonenkis dimension to obtain a learnability theorem for functions on continuous spaces.

The final section concerns connectionist networks. Specifically, we apply our dimensionality theorem to show that such networks intrinsically represent spaces of concepts that are of low dimension. We also point out that based on our results of Section 4, it is unlikely that "linear threshold" networks can be efficiently learned.

2. Preliminaries

We now describe our variant of the learning framework proposed by Valiant (1984). We will call it Framework 1, to distinguish it from those that follow.

To us, a concept is simply a subset of the objects in a predefined universe. For example, the concept of a chair is simply all objects in the world that we would call chairs. Formally, we define concepts on strings, with the understanding that these are symbolic representations for the objects in the universe of interest. Let Σ^* denote all strings on the binary alphabet $\Sigma = \{0, 1\}$. A *concept* f is any subset of Σ^*. Viewed another way, a concept is a Boolean-valued function f: $\Sigma^* \rightarrow \{0, 1\}$, where $f(x) = 1$ implies x is in the concept and $f(x) = 0$ otherwise. We will understand a concept to mean a function or a set, relying on the context to make the meaning clear.

An example for a concept is simply an object in the universe and an indication of its classification with respect to the concept. If the object is a member of the concept we call it a positive example, and if the object is not a member of the concept, a negative example. Formally, an *example* for a concept f is a pair $(x, f(x))$. If $f(x) = 1$, $(x, f(x))$ is a *positive* example, else it is a *negative* example.

Having defined what we mean by a concept and an example for it, we can define the notion of "learning a concept from examples." Informally speaking, we are concerned with the following problem: given some examples for an unknown concept and some prior information on it, compute a good approximation for the concept. Defined this way, concept learning is simply interpolation of an unknown set from a given collection of data points. As in numerical interpolation, the number of data points needed for a good approximation will depend on our prior information—for instance, whether we know the unknown function to be a polynomial of degree 3 or degree 5. We are interersted in a precise characterization of this dependence in the context of concept learning. In order to proceed further, we

need to make precise our notions of "learning," "prior information," and "good approximation," among others.

We now attempt to make precise the informal notion of "knowing something about an unknown concept." For instance, when numerically interpolating an unknown function through a given set of points, if we knew that the unknown function was a polynomial of degree 10, we would have a good handle on the interpolation task. What has our information done for us? It has served to rule out all functions that are not polynomials of degree 10, so that we need not consider polynomials of degree 20 or trigonometric functions. In some sense, prior knowledge allows us to carve out a small space[2] of concepts around the unknown concept—the space consisting of all the concepts on the universe that are consistent with the prior knowledge. We call this the space of concepts corresponding to the prior knowledge. Formally, a *space of concepts* F is any set of subsets of Σ^*.

As a first attempt, we will measure the efficacy of prior knowledge and its effect on the learning process by means of the properties of the space of concepts corresponding to it. Notice that a space of concepts has an identity independent of the prior knowledge to which it corresponds. Henceforth, we will be concerned only with spaces of concepts, although on occasion we will attempt to interpret our results in the context of prior knowledge.

Next we formalize the notion of a learner. Let F be a space of concepts on Σ^*. A *learning algorithm* for F is an algorithm that attempts to infer a concept from examples for it. The learning algorithm has at its disposal a routine EXAMPLE, which at each call produces an example for the concept to be learned. For any concept f in F, the probability that a particular example $(x, f(x))$ will be produced at any call of EXAMPLE is $P(x)$, where P is an arbitrary and unknown probability distribution on Σ^*. The choice of the distribution is independent of the concept f to be learned. (For a discussion of the intuitive significance of this probability distribution, please see the Remark below.) After seeing some examples for the unknown concept, the learning algorithm is to output the learned concept—hopefully a good approximation to the unknown concept. We will require that the concept output by the learner be consistent with the prior knowledge and hence must be in the space of concepts corresponding to the prior knowledge. One can also explore learning models where this consistency requirement can be relaxed, e.g., the notion of predictability as explored in Kearns et al. (1987a, 1987b), Haussler, Littlestone and Warmuth (1988), and Pitt and Warmuth (1988).

Suppose that in a certain learning experiment the learner sees some examples for a concept, and the length of the longest example seen is 55 characters. It might be unreasonable to expect the learner to find an approximation to the unknown concept that accurately classifies strings much longer than 55 characters. Hence, we provide as input to the learner an integer n, with the expectation that the learner will find an approximation to the concept that accurately classifies all strings of length n or less. Furthermore, the examples provided to the learner will all be of length at most n in that the probability distribution P is non-zero only on Σ^{n-},

where Σ^{n-} is the set of all strings as length n or less on Σ. In essence, we introduce n as a parameter to the learning problem and study the asymptotic behaviors of the learning algorithm as n is varied.

Since the examples provided to the learner are drawn at random, it is unreasonable to expect the learner to learn 100% of the time or be 100% accurate. Indeed, this might be possible only if the unknown concept is exhaustively enumerated. Instead, in addition to the length parameter n, we provide as input to the learner an error parameter h, with the expectation that the learner will learn with probability $(1 - 1/h)$ and that the learned approximation to the unknown concept will correctly classify any string of length n or less with probability $(1 - 1/h)$. In other words, the learner is to learn with confidence $(1 - 1/h)$ and accuracy $(1 - 1/h)$. We note here that some authors choose to use distsinct parameters to control the confidence and accuracy desired, e.g., Kearns et al. (1987b). We choose to use a single parameter in the interest of simplicity, noting that our results carry over to two distinct parameters.

Remark. In essence, the learner takes as input two integers n and h and is to approximate the unknown concept on strings of length n or less. The learner can call for examples for the concept to be learned, and these examples are chosen according to an arbitrary and unknown probability distribution P over all strings of length at most n. After seeing some number of examples, the learner outputs an approximation to the unknown concept. The approximation is to be "good" in the sense that with respect to P, with high probability the learned concept agrees with the unknown concept on strings of length n or less. By high probability, we mean probability $(1 - 1/h)$.

The significance of the probability distribution is best explained as follows. Suppose that the learner is trying to learn the concept of a sports car. He stands on a street corner in Pittsburgh and has someone point out the sports cars as they pass by. After a few days of such activity, the learner would have a reasonable idea of the concept of a sports car in that he would be able to correctly classify most cars in Pittsburgh. Here, the probability distribution P reflects the distribution of cars in Pittsburgh, the learner sees examples drawn at random according to this distribution, and his learned approximation is tested on the same distribution. The learner has learned well if his approximation correctly classifies most of the cars he sees, i.e., the probability that his approximation is incorrect is small. Of course, it is likely that the approximation the learner obtained in Pittsburgh is a bad one in Beverly Hills. But this does not reflect on the learner's ability to learn. A good learner will learn an approximation that is good in Beverly Hills, if his examples were picked from there as well.

In a different setting, suppose a student takes a course from Professor X. If the student were a good one, he should be confident of passing a test in the course so long as Professor X made up the test. On the other hand, the student can offer no guarantees on his performance in a test set by Professor Y, whose leanings on the

same material could be far from those of Professor X. However, if Y taught the course and set the test, the student should have no difficulty.

In this sense, the probability distribution P attempts to characterize the leanings of the teacher, and a good learner will succeed with high probability for any teacher, so long as the teacher and the examiner are the same entity. For additional discussion on the technical and philosophical ramifications of these assumptions, we refer the reader to Valiant (1984). Also, while we allow the examples to be chosen according to any probability distribution, one can consider fixing the distribution to a known class, as in Benedeck and Itai (1988), and Natarajan (1988a). □

Recall that the learner must output an approximation to the unknown concept and that this approximation must be in F, the space of concepts corresponding to the prior knowledge. What is the form of this output? The answer to this question depends on the focus of our inquiry, and we will consider two foci. The first inquires into the number of examples required for learning, independent of the time needed to process the examples. This is the information complexity of learning and will be the subject of study of this and the following section. The second focus inquires into the time required for learning, in the sense of the time required by the learning algorithm to process the examples. This is the time complexity of learning and will be the subject of later sections. In the student-teacher setting mentioned in the above remark, the former inquires into the amount of student-teacher interaction required for learning, whereas the latter inquires into the amount of homework the student must do to process what he has seen in class. At any rate, the important issue is that when we are interested only in the number of examples or the information complexity of learning, the form of the learning algorithm's output is immaterial. We will simply assume that the algorithm's output is the name of the concept in some predetermined naming system.

The number of examples the learner might require for length parameter n and error parameter h can depend on n and h. The nature of this dependence determines the difficulty of learning a given space of concepts. In general, if this relationship is a small polynomial, we consider the learning task to be feasible. If the relationship is super-polynomial, the number of examples required will be hopelessly large for all but small values of n and h. With this in view, we will call a space of concepts feasibly learnable if the relationship is polynomial.

Definition. Formally, a space of concepts F is *feasibly learnable* if there exists an algorithm[3] A such that

(a) A takes as input two integers n and h, where n is the size parameter and h is the error parameter.
(b) A makes polynomially few calls of EXAMPLE, polynomial in n and h. EXAMPLE returns examples for some $f \in F$, where the examples are chosen randomly and independently according to an arbitrary and unknown probability distribution P on Σ^{n-}.

(c) For all concepts $f \in F$ and all probability distributions P on Σ^{n^-}, with probability $(1 - 1/h)$, A outputs a concept $g \in F$ such that

$$\sum_{x \in f \Delta g} P(x) \leq 1/h$$

where $f \Delta g$ denotes the symmetric difference $(f - g) \cup (g - f)$.

We have now made precise our notions of "learning," "prior knowledge," "good approximation," and feasible learnability. In the following section we inquire into the properties of the prior information (more precisely, of the space of concepts) that will permit feasible learnability. This will directly address the question raised in the introduction regarding the quantitative relationship between the number of examples required for learning and the prior information.

3. Dimension and learnability

We now turn our attention to a measure of complexity for a space of concepts, which is a rather well-known measure of information complexity.

Definition. Let F be a space of concepts and let $f \in F$. The *projection* f_n of f on Σ^{n^-} is simply the set of strings of length at most n in f, i.e., $f_n = (f \cap \Sigma^{n^-})$. Similarly, the projection F_n of the space F on Σ^{n^-} is given by $F_n = \{f_n | f \in F\}$. We call F_n the n^{th}-subspace of F.

Definition. The *dimension* of subspace F_n, denoted by $dim(F_n)$ is defined by

$$dim(F_n) = log_2(|F_n|).$$

We use the notation that for a set X, $|X|$ denotes the cardinality, whereas for a string x, $|x|$ denotes the string length.

Definition. Let d:$\mathbf{N} \rightarrow \mathbf{N}$ be a function of one variable, where $\mathbf{N}$ is the natural numbers. The *asymptotic dimension* (or more simply the dimension) of a space of concepts F is $d(n)$ if $dim(F_n)$ is $\Theta d(n)$, i.e., both $O(d(n))$ and $\Omega(d(n))$. More precisely, the dimension of F is $d(n)$ if there exists a constant c such that

$$\forall\, n : dim(F_n) \leq d(n) \text{ and}$$

$$dim(F_n) \geq cd(n) \text{ infinitely often.}$$

We denote the asymptotic dimension of a space F by $dim(F)$. We say a space F is of polynomial dimension if the asymptotic dimension of F is a polynomial in n.

Example 1. For any natural number n, consider n Boolean variables $a_1, a_2 \ldots, a_n$. A monotone monomial is a Boolean formula of the form

$\bigwedge_{i \in S} a_i$, where $S \subseteq \{1, 2, \ldots, n\}$.

In words, a monotone monomial is a formula consisting solely of the conjunction of some subset of the variables $a_1, a_2 \ldots, a_n$. We can view a monotone monomial f on n variables as a concept on Σ^{n-} as follows. Let each string x of length n correspond to a truth assignment of the variables, so that the i^{th} bit of x is the value of a_i. The concept defined by f is simply all strings of length n that satisfy f. This is similar to our earlier discussion on the equivalence of Boolean-valued functions and concepts. Now consider the space F_n of all such concepts on Σ^{n-}. What is the dimension of this space? Clearly, there are only 2^n distinct monotone monomials on n variables, at most as many as there are subsets of the n variables. Hence the dimension of F_n is $log_2(2^n) = n$, and the asymptotic dimension of the space F of all monotone monomial concepts on Σ^* is n. Since this is a polynomial in n, we say it is a space of polynomial (in fact, linear) dimension. □

Hence, the dimension of a space of concepts is simply a measure of the variation of the size of the space with string length. Intuitively, the notion of dimension attempts to measure the size of the space as the complexity of the universe on which the concepts are defined is increased.

We now explore the relationship between the dimension of a space of concepts and its learnability. We first develop an interesting property of the dimension. In particular, we will show that the dimension of a space is, in some sense, the number of "degrees of freedom" of the space. (Hence the choice of the name dimension.)

Definition. Let F be a space of subsets of set X. We say that F *shatters* a set $S \subseteq X$, if for every $S_1 \subseteq S$, there exists $f \in F$ that separates S_1 from the rest of S, i.e., $f \cap S = S_1$. To our knowledge, this notion was first introduced by Vapnik and Chervonenkis (1971).

Example 2. Let F be the space of monotone monomial concepts as in Example 1. In particular, let $n = 4$. Then, the set $S = \{0111, 1011, 1101, 1110\}$ is shattered by F. To see this, pick any subset S_1 of S, say $S_1 = \{0111, 1101\}$. If f is the set determined by the formula $a_2 \wedge a_4$, then $f \in F$, and $f \cap S = S_1$. Therefore, F shatters S. □

Remark. The significance of the above definition is that if F shatters a set S of strings, then each string x in S is independent of the others with respect to F. These strings are independent in the sense that knowing whether or not $x \in F$, for an undetermined concept $f \in F$, tells us nothing about the membership in f of any other string in S. In fact, Vapnik and Chervonenkis define the dimension of a space to be the size of the largest set shattered by it and this is the definition used by Blumer et al. (1986). But then, it is often much simpler to estimate the cardinality

of a space than to estimate the size of the largest set shattered by it. For this reason, we define the dimension of a space to be the logarithm of cardinality, and as we will show shortly, the two definitions of dimension are closely related. □

We will now establish the relationship between our notion of dimension for a space of concepts and the size of the largest set shattered by the space.

Lemma 1. If F_n is of dimension d, then F_n shatters a set of size $ceiling(d/(n + 2))$, where $ceiling(r)$ is the smallest integer greater than r. Also, every set shattered by F_n is of size at most d.

Proof. Given in the Appendix. □

Remark. The significance of Lemma 1 is that if the dimension of a space is d, then there are at least $d/(n + 2)$ independent strings. Hence, we can intuitively argue that at least $d/(n + 2)$ examples would be necessary to get a good approximation of any concept in F. In fact, this argument is used formally in the proof of Theorem 1 below. □

Example 2 (continued). In Example 2, $n = 4$ and the dimension $d = n = 4$. Therefore, $ceiling(d/(n + 2)) = 1$ and hence Lemma 1 requires the existence of set of size 1 that is shattered. Note that this is a lower bound and larger sets might well be shattered. Indeed the set S of Example 2 is of cardinality 4 and is shattered by the monotone monomials. □

Lemma 1 is key to the main result of this section—a theorem relating learnability and dimension. Blumer et al. (1986) and Rivest (1987) present independent developments of variants this theorem.

Theorem 1. A space of concepts F is feasibly learnable if and only if it is of polynomial dimension.

Proof. We give an informal proof as follows.

(only if) Suppose that F were of dimension $d(n)$. Then, by Lemma 1 there exists a set of size $O(d(n)/(n + 2))$ that is shattered by F_n. As mentioned, these strings are mutually independent and hence it is not possible to predict the behavior of an unknown concept on these strings from fewer than $O(d(n)/(n + 2))$ examples. Hence, if $d(n)$ were super-polynomial in n, polynomially few examples would not suffice for sufficiently large n. For a formal proof see Blumer et al. (1986) or Natarajan (1988b).

(if) This direction of the proof follows from the claim below.

Claim. Let F be a space of concepts of dimension $d(n)$. Then, there exists a learning algorithm for F that calls for $h(d(n) + log(h))$ examples of input n, h.

Proof. We say a concept f is *consistent* with a set of examples S if for each (x, y) in S, $y = f(x)$. i.e., S is a set of examples for f. The following learning algorithm satisfies the statement of the claim.

program learn;
input: integers n, h;
begin
call EXAMPLE $h(d(n) + log(h))$ times;
let S be the set of examples seen;
pick a concept f in F consistent with S;
output f
end

In words, the algorithm above simply calls for some number of examples and picks any concept in F that is consistent with the examples thus obtained. (For this reason, learning algorithms of this form are sometimes referred to as consistent algorithms.) Although this is a simple strategy, it is provably good. A formal proof that this algorithm indeed learns F may be found in Blumer et al. (1986) or in Natarajan (1986, 1988b). The latter uses a simple counting argument, while the former uses the results of Vapnik and Chervonenkis (1971). □

To see the import of the above results, suppose the learner's prior knowledge of an unknown concept carves a space of concepts F around the concept. Then the number of examples necessary and sufficient to learn the concept is proportional to the dimension of F. We illustrate this with an example.

Example 3. Consider the n boolean variables $a_1, a_2 \ldots, a_n$. Suppose we know that the concept to be learned is a monotone monomial concept on these variables. This tells us that the unknown concept is simply one in the space F of conjunctive monomial concepts on the variables. As discussed in Example 1, the dimension of this space is $log(2^n) = n$. To learn with confidence and accuracy $(1 - 1/h)$, we simply need to call for $h(n + log(h))$ examples and pick a monotone monomial concept consistent with the examples we see. The following algorithm employs this strategy.[4]

program learnmonomial;
input: integers n, h;
begin
call for $h(n + log(h))$ examples;
let S be the examples seen;
$\Phi \leftarrow a_1 \wedge a_2 \wedge \ldots a_n$;
for each positive example $(x, 1)$ *in* S **do**
for each a_i **do**
if $a_i = 0$ in x **then** delete a_i from Φ.

```
        od
    od
    output Φ;
end
```

Notice that *learnmonomial* ignores negative examples. This is because it starts with the most restrictive monotone monomial—$a_1 \wedge a_2 \wedge \ldots a_n$, and chips away at it to accommodate the examples seen. If all the examples seen are negative, the program outputs the monomial it started with, which will be approximately correct with high probability. □

4. Time considerations in concept learning

Thus far, we have concerned ourselves with the information complexity of learning, i.e., the number of examples required to learn. Another issue to be considered is the time-complexity of learning, i.e., the time required to process the examples. In order to permit interesting measures of time complexity, we must specify the manner in which the learning algorithm identifies its approximation to the unknown concept. In particular, we will require the learning algorithm to output a name for its approximation in some predetermined naming scheme. To this end, we define the notion of an index for a space of concepts.

In order for each concept in a space F to have a name of finite length, F would have to be at most countably infinite. Assuming that the space F is countably infinite, we define an *index* of F to be a function $I{:}F \rightarrow 2^{\Sigma^*}$ such that

$$\forall f, g \in F, f \neq g \text{ implies } I(f) \cap I(g) = \varnothing.$$

For each $f \in F$, $I(f)$ is the set of indices for f.

Remark. A name for a concept is simply a string in Σ^*. The index function I maps each concept in F to a set of names. This allows for the possibility of more than one name for the same concept. □

We are primarily interested in spaces that can be learned efficiently, i.e., in time polynomial in the input parameters n, h and in the length of the shortest index for the concept to be learned. Analogous to our definition of learnability, we can now define polynomial-time learnability. Essentially, a space of concepts is polynomial-time learnable if it is feasibly learnable by a polynomial-time algorithm.

Definition. A space of concepts F is *polynomial-time learnable* in an index I if there exists a deterministic[5] learning algorithm A such that

(a) A takes as input integers n and h.

(b) A runs in time polynomial in the error parameter h, the length parameter n and in the length of the shortest index in I for the concept to be learned f. A may make polynomially few calls of EXAMPLE, polynomial[6] in n and h. EXAMPLE returns examples for some f in F, the examples being chosen randomly according to an arbitrary and unknown probability distribution P on Σ^{n-}.

(c) For all concepts f in F and all probability distributions P on Σ^{n-}, with probability $(1 - 1/h)$ the algorithm outputs an index $i_g \in I(g)$ of a concept g in F such that

$$\sum_{x \in f \Delta g} P(x) \leq 1/h.$$

We are interested in identifying the class of pairs (F, I), where F is a space of concepts and I is an index for it, such that F is polynomial-time learnable in I. Notice that in the program *learn* given in the proof of Theorem 1, the only operation that could be computationally time-consuming is that of picking a concept f in F consistent with the set of examples seen. If we can ensure that this can be done efficiently, then the learning algorithm would be provably efficient. To this end, we define the following.

Definition. For a space of concepts F and index I, an *ordering* is a program that

(a) takes as input a set of examples $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i) \ldots\}$ such that $x_1, x_2, x_3, \ldots \in \Sigma^*$, and $y_1, y_2, \ldots \in \{0, 1\}$.

(b) produces as output an index in I of a concept $f \in F$ that is consistent with S, if such exists; i.e., it outputs $i_f \in I(f)$ for some $f \in F$ such that

$$\forall (x, y) \in S, y = f(x).$$

Remark. We use the term "ordering" to describe the above notion as one can view it as a listing of the concepts in F indexed by sets of examples. The reader might prefer other names, such as "hypothesis finder" or "concept-fitting algorithm." For instance, Rivest (1987) uses the term "identification" to refer to a similar notion. □

Furthermore, if the ordering is deterministic and runs in time polynomial in the length of its input and the length of the shortest such index, we say it is a *polynomial-time ordering* and F is *polynomial-time orderable* in I. Note that we do not require the ordering to output the shortest index, only that its running time be polynomial in the length of the shortest index. Also, if the ordering is randomized and runs in polynomial time, we say F is *random polynomial-time orderable* in I.

With these definitions in hand, we can state the following theorem.

Theorem 2. A space of concepts is polynomial-time learnable in an index I (1) if it is of polynomial dimension and is polynomial-time orderable in I, and (2) only if F is of polynomial dimension and is random polynomial time orderable in I.

Proof. (If) Let Q be a polynomial-time ordering for F in I. The following is a polynomial time learning algorithm for F in I.

program learnfast;
input: n, h
begin
call EXAMPLE $h(dim(F_n) + log(h))$ times;
let S be the set of examples seen;
output $Q(S)$;
end

Given Theorem 1, we know that *learnfast* learns F, and it only remains for us bound its running time by a polynomial. Now, Q runs in time polynomial in the size of its input and the length of the shortest index of any concept consistent with S. Since the concept to be learned must be consistent with S, surely Q runs in time polynomial in n, h, and in the length of the shortest index of the concept to be learned. Hence, *learnfast* runs in time polynomial in n, h, and in the length of the shortest index for the concept to be learned. Therefore, F is polynomial-time learnable in I.

(only if) Assume that F is polynomial-time learnable in an index I by an algorithm A. Using an argument similar to that used to prove Theorem 1, we can show that F must be of polynomial dimension. For details, see Natarajan (1988b). It remains to show that there exists a randomized polynomial-time ordering for F. We give such an ordering below. The randomization technique used in this ordering is a variant of that used in the proof of the "only if" direction of Theorem 1. The same technique is used by Kearns et al. (1987) to obtain hardness results for some learning problems.

program Q;

input: S: set of examples, n: integer;
begin
place the uniform distribution on S;
let $h = |S| + 1$:
run A on inputs n, h, and
    on each call of EXAMPLE by A
    return a randomly chosen element of S;
output the index output by A;
end

Let f be a concept consistent with S whose index length is the shortest over all such concepts. Now, with probability $(1 - 1/h)$, A must output the index of a concept g such that f and g agree with probability at least $(1 - 1/h)$. Since the distribution is uniform and $h > |S|$, g must agree with f on every example in S. Hence with high probability, g is consistent with S. Furthermore, since A is a polynomial-time learning algorithm for F, our ordering Q is a randomized polynomial-time ordering for F in I. To see this, notice that A runs in time polynomial in n, h, and l, the length of the shortest index of f. By our choice of h, it follows that A runs in time polynomial in n, $|S|$, and I. Hence, Q runs in time polynomial in n, h, and l, and is a randomized polynomial-time ordering for F in I.

This completes the proof. □

Example 4. Again consider the monotone monomial concepts of Example 1. The set of monotone monomial formulae forms an index for this space of concepts. The learning algorithm *learnmonomial* of Example 3 outputs monotone monomial formulae as its approximations to the concept to be learned. Furthermore, the algorithm runs in time $nh(n + log(h))$, which is polynomial in n and h. Thus the monotone monomial concepts are polynomial-time learnable in the monotone-monomial formulae. □

5. Learning functions

Thus far our discussion has concerned concepts or sets. In the more general setting, we consider learning algorithms that learn functions from Σ^* to Σ^*. We first need to generalize our definitions.

Definition. A *space of functions* F is any set of functions from Σ^* to Σ^*.

Definition. For any $f \in F$, the projection $f_n: \Sigma^{n-} \to \Sigma^{n-}$ of f on Σ^{n-} is given by

$$f_n(x) = \begin{cases} f(x), \text{ if } |f(x)| \leq n \\ n\text{-length prefix of } f(x), \text{ otherwise} \end{cases}$$

Definition. The *n^{th}-subspace* F_n of F is the projection of F on Σ^{n-}. i.e.,

$$F_n = \{f_n | f \in F\}.$$

Remark. Intuitively, for a function f, the projection f_n attempts to capture the behavior of f with n-bit precision. If, for some string x in Σ^{n-}, $f(x)$ is of length greater than n, $f_n(x)$ is the first n bits of $f(x)$, i.e., the n-length prefix of $f(x)$. The subspace F_n of F is the analog for spaces in that F_n is the projection of F on Σ^n. □

An example for f is a pair of strings (x, y) such that $y = f(x)$. Again, we provide the learning algorithm a routine EXAMPLE that produces examples for the function to be learned, chosen randomly according to an arbitrary and unknown distribution P.

With these definitions in hand, we can formalize the notion of learnability of functions as follows.

Definition. A space of functions F is *feasibly learnable* if there exists an algorithm A such that

(a) A takes as inputs integers n and h, where n is the size parameter and h the error parameter.
(b) A makes polynomially few calls of EXAMPLE, polynomial in n and h. EXAMPLE returns examples for some function $f_n \in F_n$, the examples being chosen according to an arbitrary and unknown probability distribution P on Σ^{n-}.
(c) For all functions $f_n \in F_n$ and all probability distributions P on Σ^{n-}, with probability $(1 - 1/h)$, A outputs a function $g \in F$ such that

$$\sum_{f_n(x) \neq g_n(x)} P(x) \leq 1/h.$$

Our definition of dimension in this setting is exactly the same as the one given earlier for concepts. We now generalize the notion of shattering as follows.

Definition. Let F be a space of functions from a set X to a set Y. We say F *shatters* a set $S \subseteq X$ if there exist two functions $f, g \in F$ such that

(a) for any $s \in S$, $f(s) \neq g(s)$.
(b) for all $S_1 \subseteq S$, there exist $e \in F$ such that e agrees with f on S_1 and with g on $S - S_1$, i.e.,

$$\forall\, s \in S_1 : e(s) = f(s)$$

$$\forall\, s \in S - S_1 : e(s) = g(s).$$

We can now generalize our shattering lemma for functions as follows.

Lemma 2 (Generalized Shattering Lemma). If F_n is of dimension d, F_n shatters a set of size $ceiling(d/(3(n + 1)))$. Also, every set shattered by F_n is of size at most d.

Proof. Given in the Appendix. □

Using this lemma, we can prove the following theorem.

Theorem 3. A space of functions is feasibly learnable if and only if it is of polynomial dimension.

Proof Sketch. (if) If F_n is of dimension d, then we can use the arguments the proof of Theorem 1 to show that $h(d(n) + log(h))$ examples suffice to obtain an approximation to within $1/h$, with a confidence of $(1 - 1/h)$.

(only if) This direction of the proof uses an argument similar to that of Theorem 1, with the difference that here we employ the notion of generalized shattering and the corresponding generalized shattering lemma. □

We can also examine the time-complexity of learning a space of functions and attempt to characterize the spaces learnable in polynomial time. To this end we define the notion of an index analogous to the corresponding definition for a space of concepts.

Definition. For a space of functions F of countable cardinality, we define an index I to be a naming scheme for the functions in F, in a sense identical to that for a space of concepts.

Definition. We say a space of functions F is *polynomial-time learnable* in an index I if there exists a deterministic learning algorithm A such that

(a) A takes as input integers n and h.
(b) A runs in time polynomial in the error parameter h, the length parameter n, and in the length of the shortest index in I for the function f to be learned. A may call EXAMPLE polynomially few times, polynomial in n and h. EXAMPLE returns examples for f_n, the examples being chosen randomly according to an arbitrary and unknown probability distribution P on Σ^{n^-}.
(c) For all functions f in F and all probability distributions P on Σ^{n^-}, with probability $(1 - 1/h)$ the algorithm outputs an index $i_g \in I(g)$ of a function g in F such that

$$\sum_{f_n(x) \neq g_n(x)} P(x) \leq 1/h.$$

We are interested in identifying the class of pairs (F, I), where F is a space of functions and I is an index for it, such that F is polynomial-time learnable in I. To this end, we define the following.

Definition. For a space of functions F and index I, an *ordering* is a program that

(a) takes as input a set of examples $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots\}$. Let n be the length of the longest string among the x_i and y_i.
(b) produces as output an index in I of a function $f \in F$ that is consistent with S, if such exists; i.e., it outputs $i_f \in I(f)$ for some $f \in F$ such that

$$\forall\ (x, y) \in S,\ y = f_n(x).$$

Furthermore, if the ordering runs in time polynomial in the length of its input and the length of the shortest such index, we say it is a *polynomial-time ordering* and F is *polynomial-time orderable* in I. If the ordering runs in random-polynomial time, we say F is *random-polynomial-time orderable* in I.

With these definitions in hand, we can state the following theorem.

Theorem 4. A space of functions is polynomial-time learnable: (1) if it is of polynomial dimension and polynomial-time orderable, and (2) only if it is of polynomial dimension and is orderable in random polynomial time.

Proof. Similar to that of Theorem 3. □

6. Learning from examples and background information

In this section, we consider general-purpose learning programs that may be used over many domains. Specifically, such programs take as input a description of the space of functions to be learned and, after some precomputation, behave like learning algorithms for that space. We will refer to such algorithms as general purpose learning programs.

Consider a general-purpose learning program M that works over a set of spaces $G_1, G_2, \ldots, G_i, \ldots$; i.e., M takes as input the description of some G_i and then behaves as a learning algorithm for G_i. Now, if $G = G_1 \cup G_2 \ldots \cup G_i \ldots$ is itself a space of low dimension, then it follows from Theorem 1 that we can build a learning algorithm for G and not bother with the complications of M. The interesting question is whether it is possible for G to be of intractably-high dimension and yet be decomposable into $G_1 \ldots, G_i \ldots$ such that each G_i is of low dimension and each G_i has a short description that can be fed into M. Such decomposability would be desirable, as it would permit the construction of a general-purpose learning algorithm M for G, so that even though G is of high dimension and difficult to learn, M would efficiently handle all the components of G, given their descriptions. In order to explore this further, we consider the Framework 2 presented below.

Let F be a space of functions. A *theory* for F is simply any total function from F to Σ^*. The *theory string* for a function $f \in F$ is the string $t = T(f)$.

Remark. Consider the typical explanation based learning framework (e.g., Mitchell, Keller & Kedar-Cabelli, 1986). When attempting to teach an explanation based learning system a new concept, it is necessary to load in the relevant domain theory and then provide it with examples. The domain theory reflects some prior information on the concept to be learned and, typically, different concepts will require different domain theories. Our notion of theory attempts to capture this idea. □

A learning algorithm for F is an algorithm that attempts to infer functions in F from examples and background information. The learning algorithm has at its disposal a routine TEACHER, which is best described as the pair {EXAMPLE, T}, where EXAMPLE is the source of random examples described in Framework 1 and T is a theory for F. When attempting to teach the learning algorithm a function $f \in F$, TEACHER behaves thus: On the first call, TEACHER returns $t = T(f)$. On subsequent calls, TEACHER returns a randomly chosen example for the projection f_n of f by invoking EXAMPLE.

Remark. We are interested in the relationship between the length of a theory string and the number of examples required for learning before and after reading the theory string. In some sense, this represents the tradeoff between background information and the information obtained from examples. Is it possible that a single bit of theory is worth many examples? We explore this below. □

Definition. We say that a space of functions F is *feasibly learnable* in Framework 2 if there exists a learning algorithm A and a theory T for F such that

(a) A takes as input integers n and h.
(b) A makes polynomially many calls of TEACHER = {EXAMPLE, T}, polynomial in n and h. A reads the first l bits of the theory string returned by TEACHER, where l varies polynomially in n and h.
(c) For all functions f in F, and all probability distributions P over Σ^{n-}, with probability $(1 - 1/h)$ the algorithm A outputs a function g in F such that

$$\sum_{f_n(x) \neq g_n(x)} P(x) \leq 1/h.$$

We are now ready to state our result.

Theorem 5. A space of functions is feasibly learnable in Framework 2 if and only if it is of polynomial dimension.

Proof. Given in the Appendix. □

Corollary. A space of functions is feasibly learnable in Framework 2 if and only if it is feasibly learnable in Framework 1.

This answers our question at the beginning of this section: If G is a space of high dimension, then G is not decomposable into component spaces of low dimension with short descriptions. Does this mean that general-purpose learning programs are not very useful? It does not, for two reasons.

First, note that we are comparing the number of examples used by the learning algorithm with the number of bits of theory read by it. If each example is of length n, where n is the length parameter, this compares the information from examples

and theory in a bit-for-bit fashion, within the factor of n. In some situations such a comparison may not be meaningful, as the cost of each example might be significantly higher than the cost of each bit of theory.

Second, if $NP \neq RP$[7], there exist spaces of functions that are polynomial-time learnable in Framework 2, but not polynomial time learnable in Framework 1. Essentially, a space of functions is polynomial-time learnable in Framework 2 if it is feasibly learnable in Framework 2 by a polynomial-time algorithm. Formally, a space F of functions is *polynomial-time learnable* in an index I for F in Framework 2 if there exists a theory T for F and a deterministic learning algorithm A such that

(a) A takes as input integers n and h.
(b) A runs in time polynomial in n, h and in the length of the shortest index for the function to be learned f. A may call TEACHER polynomially few times, polynomial in n and h. A reads the first l bits of the theory string returned by TEACHER, where l varies polynomially in n and h.
(c) For all functions f in F, and for all probability distributions P over Σ^{n-}, with probability $(1 - 1/h)$ the algorithm outputs an index $i_g \in I(g)$ of a function g in F such that

$$\sum_{x \in f_n(x) \neq g_n(x)} P(x) \leq 1/h.$$

Theorem 6. If $NP \neq RP$, there exists a space of functions F and an index I for it, such that F is polynomial-time learnable in I in Framework 2, but not in Framework 1.

Proof Sketch. The proof rests on a result of Kearns et al. (1987) concerning the existence of spaces that are of polynomial dimension but not polynomial-time learnable in Framework 1, under the assumption $NP \neq RP$.

A function f is a Boolean threshold function on n variables $a_1, a_2, \ldots, a_n$ if there exists $S \subseteq \{a_1, a_2, \ldots, a_n\}$ and $1 \leq k \leq n$ such that

$$f(x) = \begin{cases} 1, \text{ if at least } k \text{ of the variables in } S \text{ are set to 1 in } x \\ 0, \text{ otherwise} \end{cases}$$

where x is a Boolean assignment to the variables $a_1, a_2 \ldots, a_n$. Kearns et al. (1987) show that the space of Boolean threshold functions is not polynomial-time learnable unless $RP = NP$. Yet it is easy to show that this space of functions is polynomial-time learnable in Framework 2. To do so, we construct a theory function T for the space, where the theory string for each function is a complete description of the function in that it enumerates the set S and the value of k. □

The significance of Theorem 6 is that, although the availability of theory cannot reduce the information complexity of learning, it can reduce the computational

complexity of learning. In the proof of Theorem 6, this reduction was obtained in a trivial way in that the theory string for each function gave the learner a complete description of the function. Natarajan (1987) considers the possibility of non-trivial reductions, but without significant progress in that direction.

Theorem 5 might seem contrary to the intuition of some readers. We urge the reader to bear in mind that the theorem does not claim that theory is useless, but merely that theory cannot offer super-polynomial reductions in information complexity. In the proof of Theorem 5, we show that the length of a theory string is comparable to the reduction in the dimension of the space of functions achieved by it. In interpreting this, we should note that a theory string does not require repeated sampling to establish confidence and accuracy. Hence, if the confidence and accuracy required is one part in one hundred, each bit of theory could be worth one hundred times as many examples.

7. Finite learnability

Thus far we have explored the asymptotic learnability of spaces of sets and functions, that is, we have considered the asymptotic variation of the number of examples needed for learning with increasing values of the size parameter. We will now investigate a different notion of learnability, one that asks whether the number of examples needed for learning is finite, i.e., varies as a finite-valued function of the error parameter, without regard to the size parameter. We call this "finite learnability" as opposed to the notion of asymptotic learnability.

For the case of spaces of sets, Blumer et al. (1986) present conditions necessary and sufficient for finite learnability. Their elegant results rely on the powerful results in classical probability theory of Vapnik and Chervonenkis (1971). In this section, we review their results briefly and then go on to present learnability results for spaces of functions, relying in part on the same results of Vapnik and Chervonenkis.

Definition. Let F be a space of concepts on $\mathbf{R}^k$, where $\mathbf{R}$ is the set of reals and k is fixed natural number. We say F is *finitely-learnable* if there exists an algorithm A such that

(a) A takes as input an integer h, the error parameter.
(b) A makes finitely many calls of EXAMPLE, although the exact number of calls may depend on h. EXAMPLE returns examples for some function f in F, the examples being chosen randomly according to an arbitrary and unknown probability distribution P on $\mathbf{R}^k$.
(c) For all probability distributions P and all functions f in F[8], with probability $(1 - 1/h)$, A outputs $g \in F$ such that

$$\int_{f \neq g} dP \leq 1/h.$$

The following theorem is adapted from Blumer et al. (1986).

Theorem 7. A space of sets F on $\mathbf{R}^k$ is finitely learnable if and only if there exists a finite bound on the size of the subsets of $\mathbf{R}^k$ shattered by F. Blumer et al. (1986) refer to the size of the largest set shattered by F as the *Vapnik-Chervonenkis dimension* of the space F.

Let us now formalize the notion of finite learnability of spaces of functions on the reals.

Definition. Let F be a space of functions from $\mathbf{R}^k$ to $\mathbf{R}^k$. We say F is *finitely-learnable* if there exists an algorithm A such that

(a) A takes as input an integer h, the error parameter.
(b) A makes finitely many calls of EXAMPLE, although the exact number of calls may depend on h. EXAMPLE returns examples for some function f in F, where the examples are chosen randomly according to an arbitrary and unknown probability distribution P on $\mathbf{R}^k$.
(c) For all probability distributions P and all functions f in F, with probability $(1 - 1/h)$, A outputs $g \in F$ such that

$$\int_{f \neq g} dP \leq 1/h.$$

We need the following supporting definitions.

Definition. Let f be a function from $\mathbf{R}^k$ to $\mathbf{R}^k$. We define the *graph set* of f, denoted by *graph*(f), to be the set of all examples for f. That is,

$$graph(f) = \{(x, y) | y = f(x)\}.$$

Clearly, $graph(f) \subseteq \mathbf{R}^k \times \mathbf{R}^k$. Analogously, for a space of functions F, we define the *graph space*, denoted by *graph*(F), to be the space of graphs sets of the functions in F. That is,

$$graph(F) = \{graph(f) | f \in F\}.$$

We now state the main theorem of this section. The theorem is not tight in the sense that the necessary and sufficient conditions do not match. Natarajan and Tadepalli (1988) reported a tight version of the theorem on the basis of an incorrect proof. More recently, Natarajan (1988b) identifies a space of functions that is finitely learnable and sits in the gap between these conditions. We note that while there are convergence results in the literature for functions (Pollard, 1986) that are akin to those of Vapnik and Chervonenkis (1971) for sets, these results do not resolve

the gap in Theorem 8. This is because these results concern uniform convergence properties and not the information complexity properties required for our purposes.

Theorem 8. A space of functions F from $\mathbf{R}^k$ to $\mathbf{R}^k$ is finitely-learnable

(a) If there exists a bound on the size of the sets in $\mathbf{R}^k \times \mathbf{R}^k$ shattered by the space *graph*(F). (This is simple shattering as defined in Section 2.)
(b) Only if there exists a bound on the size of the sets in $\mathbf{R}^k$ shattered by F. (This is generalized shattering as defined in Section 5.)

Proof sketch. (If) This direction of the proof follows from the convergence results of Vapnik and Chervonenkis (1971), exactly as in Blumer et al. (1986). Essentially, the "if" condition implies that the space *graph*(F) is finitely learnable. It follows that the space F is finitely learnable.

(Only if) This direction of the proof is identical to that of Theorem 3, which in turn followed the arguments of Theorem 1. In essence, the size of the set that is shattered by F is the number of independent pieces of information necessary to identify a function in F. If this is unbounded, then F will not be learnable from finitely many examples. □

In the foregoing, we considered functions on real spaces, requiring that on a randomly chosen point, with high probability, the learner's approximation agree exactly with the target function. In a sense, this result is largely of technical interest, as practical computations can only be carried out to some finite degree of precision. Of course, if all the computations were carried out to some finite precision, Theorem 3 would apply directly. Alternatively, we could demand that the learned function approximate the target function with respect to some standard norms, say the square norm. Specifically, we could replace the integral in condition (c) of the definition of a finitely learnable class of functions with the norm of interest. In the case of the square norm, it would read

$$\int_{x \in \mathbf{R}^k} E(f(x), g(x)) dP \leq 1/h,$$

where $E(a, b)$ is the Euclidean distance between a and b in $\mathbf{R}^k$. For such norms, we can prove the equivalent of Theorem 8 for the case where P is a fixed distribution. Such a theorem involves the use of covers as defined in Benedeck and Itai (1988). In brief, we define a $(1/h)$-cover of F, with respect to the square norm and the distribution P, to be a subset K of F such that for each $f \in F$ there exists $g \in K$ such that f and g satisfy the above inequality. Then, we can show that F is finitely learnable over P if for all h, there exists a finite $(1/h)$-cover for F. This is discussed in greater detail in Natarajan (1989b).

8. An application to connectionist networks

In this section we discuss an application of our results to connectionist networks. These networks are the subject of numerous investigations (Rumelhart & McClelland, 1986) and appear to be promising as machine architectures for learning. In the following, we will show that such networks intrinsically represent spaces of functions that are of low dimension.

A *connectionist network* (more simply, a network) consists of a set of nodes and a set of input and output terminals. Each node has several input wires and one output. The input wires may be connected to the output points of other nodes or to the terminals that serve as the input to the entire network. Similarly, the output of a node may connect to inputs of other nodes or to the output terminals of the entire network. Each input wire of a node has a weight associated with it. Each node computes the same function—the *node function*—$\tau: \mathbf{R} \to \mathbf{R}$ and outputs the value of τ on the sum of its inputs duly scaled by the appropriate weights. More precisely, for a certain node, let x_i and w_i be the i^{th} input and weight, respectively. The value of the output of the node is $\tau(s)$, where $s = \sum_{\text{all inputs}} w_i \cdot x_i$.

These networks are used to compute functions as follows. Signals of (0, 1) value are applied to the input terminals of the network. The nodes connected to these terminals change their output values based on these signals and the effect propagates through the network. The output terminals of the network display new values as a result of the propgataion. (If the network has cycles in it, we consider the values of the output terminals when the network equilibrates, if at all.) The values at the output terminals form the value of the function computed by the network for the applied input. The function computed by the network can be changed by altering the weights on the inputs of the nodes. Depending on the range of the weights, a network can be configured to compute any of a space of functions. Thus it can be used to learn a function from this space by picking examples for the function to be learned and then changing the weights on the network so that the network agrees with the examples. Additional details and background on such networks may be found in Rumelhart and McClelland (1986).

We are interested in the following question. What is the dimension of the space of functions that can be computed by a network of n nodes? As it happens, we can provide an upper bound on the dimension, independent of the node function used by the nodes.

Consider a network of n nodes where each node is connected to the outputs of all the other $n - 1$ nodes. Let each weight be a b-bit binary integer. Rather than specify input and output terminals, we will use the following convention. We designate those nodes that have zero weights on all their input wires to be input terminals. Also, we assume n output terminals, where each is connected to the output of a single node. It is easy to see that this convention loses no generality with respect to the functions computed by these networks. With these definitions behind us, we estimate the number of different functions that a network of n nodes

can compute. There are at most $n(n-1)$ weights, and each weight is a binary integer of b bits. This implies that there are at most $2^{bn(n-1)}$ distinct weight assignments to the network and hence at most as many functions computed by it. Hence, the space of functions computable by the network is of cardinality at most $2^{bn(n-1)}$ and therefore of dimension at most $bn(n-1)$.

Thus, connectionist networks intrinsically support spaces of functions that are of dimension polynomial in the size of the network. Since the length of the input is no more than the size of the network, the space of functions represented by the network is feasibly learnable by Theorem 1. Unfortunately, most interesting node functions investigated to date generate networks that are not known to be efficiently orderable and, therefore, by Theorem 2, not polynomial-time learnable. Specifically, given a set of examples for the function to be learned, it is computationally intractable to adjust the weights in the network to obtain agreement between function and network. See Blum and Rivest (1988) for more on this issue.

In the foregoing, we considered networks with finite precision integer weights. What if the weights were allowed to be real numbers of arbitrary precision? If we allow infinite precision weights, we must limit the class of node functions to keep things meaningful. Otherwise, the network can compute any function of its inputs and thereby support spaces of functions of arbitrary dimension. Consider the *linear threshold function* τ, defined as follows.

$$\tau(s) = \begin{cases} 1 \text{ if } s \geq 1 \\ 0 \text{ otherwise} \end{cases}$$

We refer to networks with linear threshold node functions as *linear threshold networks*. The interesting thing here is that a linear threshold network of n nodes with arbitrary precision weights can be replaced exactly by a linear threshold network of n nodes with $(n\log n)$-bit integer weights, (J-W. Hong, personal communication, 1987). Hence, our arguments on the dimension of the space of functions represented by a network of n nodes hold for networks with linear threshold functions, even if the weights are arbitrary real numbers. In particular, a linear threshold function network of n nodes supports a space of dimension at most $n^2(n-1)log(n)$, even if the weights are arbitrary real numbers.

If a connectionist network is good model of the human brain, then the space of functions supported by the one would be a good model of the space of functions supported by the other. It would follow that a connectionist network is a good architecture for learning those functions that are learned easily by the human brain. The import of this section is that the types of networks (such as the linear threshold network) investigated to date are efficient with respect to the number of examples required for learning, but not with respect to the computational time involved. The difficulty lies in ordering them, i.e., given a set of examples for a network, it is computationally difficult to adjust the weights in the network so that the network agrees with the examples. Given this, one might wonder whether these types of

networks are too powerful and not very good models of the human brain. It would be interesting to investigate other types of networks that are efficiently orderable, even if they are computationally less powerful than the types of networks that are the subject of current investigations.

9. Conclusion

This paper concerned learning concepts and functions. Two frameworks were defined, one that provided the learner with examples for the unknown concept or function, and another that provided the learner with examples and background information. Within the first framework, we identified necessary and sufficient conditions for learning from a small number of examples and in a time-efficient manner. As for the second framework, we showed that the addition of background information gave this framework no advantage over the first with respect to information complexity, although it might offer advantages with respect to time complexity. We also considered an application of our results to connectionist networks to show that these networks intrinsically support hypothesis spaces of low dimension.

Another possible application of these results is in the context of learning heuristics for solving puzzles and problems such as symbolic integration. This is explored by Natarajan and Tadepalli (1988) and by Natarajan (1989a). The first paper applies some of the results of this paper towards learning deterministic heuristics, given solved sample instances from a problem domain. The second paper discusses the learning of probabilistic heuristics from solved instances, as well as from unsolved instances or exercises over a problem domain.

In all, this paper presented some theoretical results on learning sets and functions, attempting to link the assumptions and results to their counterparts in the artificial intelligence literature.

Appendix

This appendix is a collection of technical proofs to some of the results in the body of the paper.

Lemma 1. If F_n is of dimension d, then F_n shatters a set of size[9] $ceiling(d/(n + 2))$. Also, every set shattered by F is of size at most d.

Proof (From Natarajan, 1988b). First, we prove the upper bound. Suppose a set S is shattered by by F_n. Since there are $2^{|S|}$ distinct subsets of F_n, it follows from the definition of shattering that $2^{|S|} \leq |F_n|$. Taking logarithms on both sides of the inequality, we get $|S| \leq log(|F_n|) = d$, which is as desired.

We prove the lower bound part of the lemma through the following claim. A variant of the claim is given by Vapnik and Chervonenkis (1971), among others.

Claim. Let X be any finite set and let H be a set of subsets of X. If k is the size of the largest subset of X shattered by H, then

$$|H| \leq (|X| + 1)^k.$$

Proof. By induction on $|X|$, the size of X.

Basis. Clearly true for $|X| = 1$.

Induction. Assume the claim holds for $|X| = m$ and prove true for $m + 1$. Let $|X| = m + 1$ and let H be any set of subsets of X. Also, let k be the size of the largest subset of X shattered by H. Pick any $x \in X$ and partition X into two sets $\{x\}$ and $Y = X - \{x\}$. Define H_1 to be the set of all sets in H that are reflected about x. That is, for each set h_1 in H_1, there exists a set $h \in H$ such that h differs from h_1 only in that h does not include x. Formally,

$$H_1 = \{h_1 | h_1 \in H, \exists\, h \in H, h \neq h_1 \text{ and } h_1 = h \cup \{x\}\}.$$

Now define $H_2 = H - H_1$. Surely, the sets of H_2 can be distinguished on the elements of Y. That is, no two sets of H_2 can differ only on x, by virtue of our definition of H_1. Hence, we can consider H_2 as sets defined on Y. Surely, H_2 cannot shatter a set larger than the largest set shattered by H. Hence, H_2 shatters a set no bigger than k. Since $|Y| \leq m$, by the inductive hypothesis we have $|H_2| \leq (|Y| + 1)^k$.

Now consider H_1. By definition, the sets of H_1 are all distinct on Y. That is, for any two distinct sets h_1, h_2, in H_1, $h_1 \cap Y \neq h_2 \cap Y$. Suppose H_1 shattered a set $S \subseteq Y$, $|S| \geq k$. Then, H would shatter $S \cup \{x\}$. But, $|S \cup \{x\}| \geq k + 1$, which is impossible by assumption. Hence, H_1 shatters a set of at most $(k - 1)$ elements in Y. By the inductive hypothesis, we have

$$|H_1| \leq (|Y| + 1)^{k-1}.$$

Combining the two bounds, we have

$$\begin{aligned} |H| &= |H - H_1| + |H_1| = |H_2| + |H_1| \\ &\leq (|Y| + 1)^k + (|Y| + 1)^{k-1} \leq (m + 1)^k + (m + 1)^{k-1} \\ &\leq (m + 1)^{k-1}(m + 2) \leq (m + 2)^k \leq (|X| + 1)^k. \end{aligned}$$

Thus the claim is proved. □

Returning to the lemma, we see that if X is all strings of length n or less on the binary alphabet, then $|X| = 2^{n+1}$. By our claim, if the largest set shattered by F_n is of size k,

$$|F_n| \leq (2^{n+1} + 1)^k.$$

Hence, $k \geq log(|F_n|)/log(2^{n+1} + 1) \geq dim(F_n)/(n + 2)$.

Since k must be an integer, we take the ceiling of the right-hand side of the last inequality. This completes the proof of the lemma. □

Lemma 2 (Generalized Shattering Lemma). If F_n is of dimension d, F_n shatters a set of size $ceiling(d/(3n + 3))$. Also, every set shattered by F_n is of size at most d.

Proof. (From Natarajan, 1988b). The upper bound part of the lemma can be proved exactly as the corresponding part of Lemma 1. To see that this upper bound can be attained, we simply need to consider a family F_n of {0, 1}-valued functions.

The lower bound part of the lemma is proved through the following claim.

Claim. Let X and Y be two finite sets and let H be a set of functions from X to Y. If k is the size of the largest subset of X shattered by H, then

$$|H| \leq (|X|)^k(|Y|)^{2k}.$$

Proof. By induction on $|X|$.

Basis. Clearly true for $|X| = 1$, for all $|Y|$.

Induction. Assume true for $|X| = l$, $|Y| = m$, and prove true for $|X| = l + 1$, $|Y| = m$. Let $X = \{x_1, x_2, \ldots, x_l\}$ and $Y = \{y_1, y_2, \ldots, y_l\}$. Define the subsets H_i of H as follows:

$$H_i = \{f | f \in H, f(x_1) = y_i\}.$$

Also, define the sets of functions H_{ij} and H_0 as follows:

$$\text{for } i \neq j\text{: } H_{ij} = \{f | f \in H_i, \exists\, g \in H_j \text{ such that } f = g \text{ on } X - \{x_1\}\}.$$

$$H_0 = H - \bigcup_{i \neq j} H_{ij}.$$

Now,

$$|H| = |H_0| + |\bigcup_{i \neq j} H_{ij}| \leq |H_0| + \sum_{i \neq j} |H_{ij}|.$$

We seek bounds on the quantities on the right-hand side of the last inequality. By definition, the functions in H_0 are all distinct on the m elements of $X - \{x_1\}$. Furthermore, the largest set shattered in H_0 must be of cardinality no greater than k. Hence, we have by the inductive hypothesis,

$$|H_0| \leq l^k m^{2k}.$$

And then, every H_{ij} shatters a set of cardinality at most $k - 1$, as otherwise H would shatter a set of cardinality greater than k. Also, since the functions in H_{ij} are all distinct on $X - \{x_1\}$, we have by the inductive hypothesis,

$$\text{For } i \neq j, |H_{ij}| \leq l^{k-1} m^{2(k-1)}.$$

Combining the last three inequalities, we have

$$\begin{aligned} |H| &\leq l^k m^{2k} + \sum_{i \neq j} l^{k-1} m^{2(k-1)} \leq l^k m^{2k} + m^2 l^{k-1} m^{2(k-1)} \leq l^k m^{2k} + l^{k-1} m^{2k} \\ &\leq l^{k-1} m^{2k} (l + 1) \leq (l + 1)^k m^{2k}. \end{aligned}$$

Which completes the proof of the claim. □

Returning to the lemma, we have $X = Y = \Sigma^{n^-}$, and hence $l = m = 2^{n+1}$. If k is the cardinality of the largest set in Σ^{n^-} shattered by F_n, we have by our claim,

$$\begin{aligned} |F_n| &\leq (2^{n+1})^k (2^{n+1})^{2k} \\ &\leq 2^{k(3n+3)}. \end{aligned}$$

Taking logarithms, we have

$$log(|F_n|) = dim(F_n) = d \leq k(3n + 3)$$

Hence, $k \geq d/(3n + 3)$, which is as desired. □

Theorem 5. A space of functions F is learnable in Framework 2 if and only if F is of polynomial dimension.

Proof. (From Natarajan (1988a)) Informally speaking, the proof of this theorem is almost obvious in the sense that a simple counting argument lies at its heart. However, a formal proof requires the argument to be buried in detail.

Abusing notation, we extend the theory function T to subsets of F as follows:

$$\text{For } B \subseteq F, T(B) = \{T(f) | f \in B\}.$$

Also, we define the inverse function T^- from Σ^* to subsets of F as given below:

For $t \in \Sigma^*$, $T^-(t) = \{f | f \in F, t \text{ is a prefix of } T(f)\}$

(if) By Theorem 1, if F is of polynomial dimension, then F is learnable in Framework 1. Hence it is learnable in Framework 2, as Framework 1 is but a special case of Framework 2.

(only if) Let A be a learning algorithm for F in Framework 2 using a TEACHER = {EXAMPLE, T} for some theory T of F. For any n and h, let $T_{n,h}$ be the set of theory strings read by A over all functions in F. These will be prefixes (of length at most l) of the theory strings offered by TEACHER over all functions in F. For any string t, let $F_n(t)$ denote the n^{th}-subspace of $T^-(t)$. Also, let

$$d_{max} = max \{dim(F_n(t)) | t \in T_{n,h}\}, \text{ and}$$

$$l_{max} = max \{length(t) | t \in T_{n,h}\}.$$

Claim. $d_{max} + l_{max} \geq dim(F_n)$.

Proof. By a simple counting argument. Since $T_{n,h}$ is over all the functions in F, surely

$$F_n \subseteq \bigcup_{t \in T_{n,h}} F_n(t).$$

Hence,

$$|F_n| \leq \left|\bigcup_{t \in T_{n,h}} F_n(t)\right| \leq \sum_{t \in T_{n,h}} 2^{dim(F_n(t))} \leq 2^{l_{max}} 2^{d_{max}} \leq 2^{l_{max} + d_{max}}$$

Thus, $dim(F_n) \leq l_{max} + d_{max}$.

From the above claim, if $dim(F_n)$ is super-polynomial in n, then either l_{max} or d_{max} varies superpolynomially with n. The former is a contradiction, since A reads only polynomially many bits of the theory strings. The latter is a contradiction as well, using the arguments of the proof of Theorem 1. Hence F is of polynomial dimension. This completes the proof. □

Acknowledgements

My thanks to T. Mitchell and P. Tadepalli for giving freely of their time and patience, and to P. Langley for reading various drafts of my papers and guiding me towards versions more appropriate to this journal. My deepest thanks to the reviewers, who contributed immensely to this paper.

Notes

1. We mean worst case with respect to the assumptions regarding the source of examples for learning, not worst-case complexity analysis.
2. Our use of the terms "space of concepts" and "space of functions" is equivalent to the term "hypothesis space" in the AI literature and the term "family" in the formal literature.
3. Unless stated otherwise, by "algorithm" we mean a finitely representable procedure, not necessarily computable. That is, the procedure might use well-defined but non-computable functions as primitives.
4. Valiant (1984) uses the same algorithm to learn conjunctive normal form expressions with a bounded number of clauses.
5. We could equally well allow for randomized algorithms, obtaining the corresponding results in a similar fashion. A randomized algorithm is one that tosses coins during its computation and produces the correct answer with high probability, Gill (1977).
6. Alternatively, we could allow as many calls of EXAMPLE as possible within the time limit. As this will not change our discussion substantially, we avoid this alternative in the interest of clarity.
7. *NP* and *RP* are the classes of functions computable in non-deterministic polynomial time and random polynomial time respectively.
8. F should satisfy some "niceness" conditions as in Vapnik and Chervonenkis (1971) and Blumer et al. (1986).
9. The expression *ceiling*(r) denotes the least integer greater than r.

References

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

Angluin, D. (1986). *Learning regular sets from queries and counter-examples* (Technical Report YALEU/DCS-464). New Haven, CT: Yale University, Department of Computer Science.

Angluin, D. & Laird, P. (1968). Learning from noisy examples. *Machine Learning, 2,* 343–370.

Benedeck, G. M., & Itai, N. (1988). Learning by fixed distributions. *Proceedings of the Workshop on Computational Learning Theory* (pp. 80–90). Cambridge, MA: Morgan Kaufmann.

Berman, P., & Roos, R. (1987). Learning one-counter languages in polynomial time. *Proceedings of the Symposium on Foundations of Computer Science* (pp. 61–67). Los Angeles, CA: IEEE Computer Society Press.

Blum, A., & Rivest, R. (1988). Training a 3-node neural network is NP-complete. *Proceedings of the Workshop on Computational Learning Theory* (pp. 9–18). Cambridge, MA: Morgan Kaufmann.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1986). Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. *Proceedings of the ACM Symposium on Theory of Computing* (pp. 273–282). Berkeley, CA: ACM Press.

Gill, J. (1977). Computational complexity of probabilistic turing machines. *SIAM Journal of Computing, 6,* 675–695.

Haussler, D., Littlestone, N., & Warmuth, M. (1988). Predicting {0, 1} functions on randomly drawn points. *Proceedings of the Workshop on Computational Learning Theory* (pp. 280–296). Cambridge, MA: Morgan Kaufmann.

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

Kearns, M., Li, M., Pitt, L., & Valiant, L. G. (1987b). Recent results on Boolean concept learning *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 337–352). Irvine, CA Morgan Kaufmann.

Laird, P. (1987). *Learning from data good and bad.* Doctoral Dissertation, Department of Computer Science, Yale University, New Haven, CT.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound. *Machine Learning, 2,* 285–318.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based generalization: A unifying view. *Machine Learning, 1,* 47–80.

Natarajan, B. K. (1986). *On learning Boolean functions* (Technical Report CMU-RI-TR-86-17). Pittsburgh, PA: Carnegie Mellon University, Robotics Institute. Also *Proceedings of the ACM Symposium on Theory of Computing, 1987* (pp. 296–304). New York, NY: ACM Press.

Natarajan, B. K. (1987). *Two new frameworks for learning* (Technical Report CMU-RI-TR87-25). Pittsburgh, PA: Carnegie Mellon University, Robotics Institute.

Natarajan, B. K. (1988a). Learning over classes of distributions. *Proceedings of the Workshop on Computational Learning Theory.* Cambridge, MA: Morgan Kaufmann.

Natarajan, B. K. (1988b). Some results on learning. Unpublished manuscript.

Natarajan, B. K. (1989a). *On learning from exercises* (Technical Report CMU-RI-TR4-89). Pittsburgh, PA: Carnegie Mellon University, Robotics Institute.

Natarajan, B. K., *Probably approximate learning over classes of distributions* (Technical Report HPL-SAL-89-29). Palo Alto, CA: Hewlett Packard Research Laboratories.

Natarajan, B. K., & Tadepalli, P. (1988). Two new frameworks for learning. *Proceedings of the Fifth International Symposium on Machine Learning* (pp. 402–415). Ann Arbor, MI: Morgan Kaufmann.

Pitt, L., & Warmuth, M. (1988). Reductions among prediction problems: On the difficulty of predicting automata. *Proceedings of 3rd IEEE Conference on Structure in Complexity Theory* (pp. 60–69). Washington, D.C.: ACM Press.

Pollard, J. (1986). *Convergence of stochastic processes.* New York: Springer-Verlag.

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

Rivest, R., & Schapire, R. E. (1987). Diversity based inference of finite automata. *Proceedings of the Symposium on Foundations of Computer Science* (pp. 78–87). Los Angeles, CA: IEEE Computer Society Press.

Rumelhart, D., & McClelland, J. (Eds.). (1986). *Parallel distributed processing.* Cambridge, MA: MIT Press.

Valiant, L. G. (1984). A theory of the learnable. *Proceedings of the ACM Symposium on Theory of Computing* (pp. 436–445). Washington, D.C.: ACM Press.

Vapnik, V. N., & Chervonenkis, A.Ya. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of probability and its applications, 16,* 264–280.