

7N-63-71N

91472

P-45

N92-70673

Unclas  
0091472

29/63

(NASA-TM-107912) A SURVEY OF COMPUTATIONAL  
LEARNING THEORY (NASA) 45 P

# A Survey of Computational Learning Theory

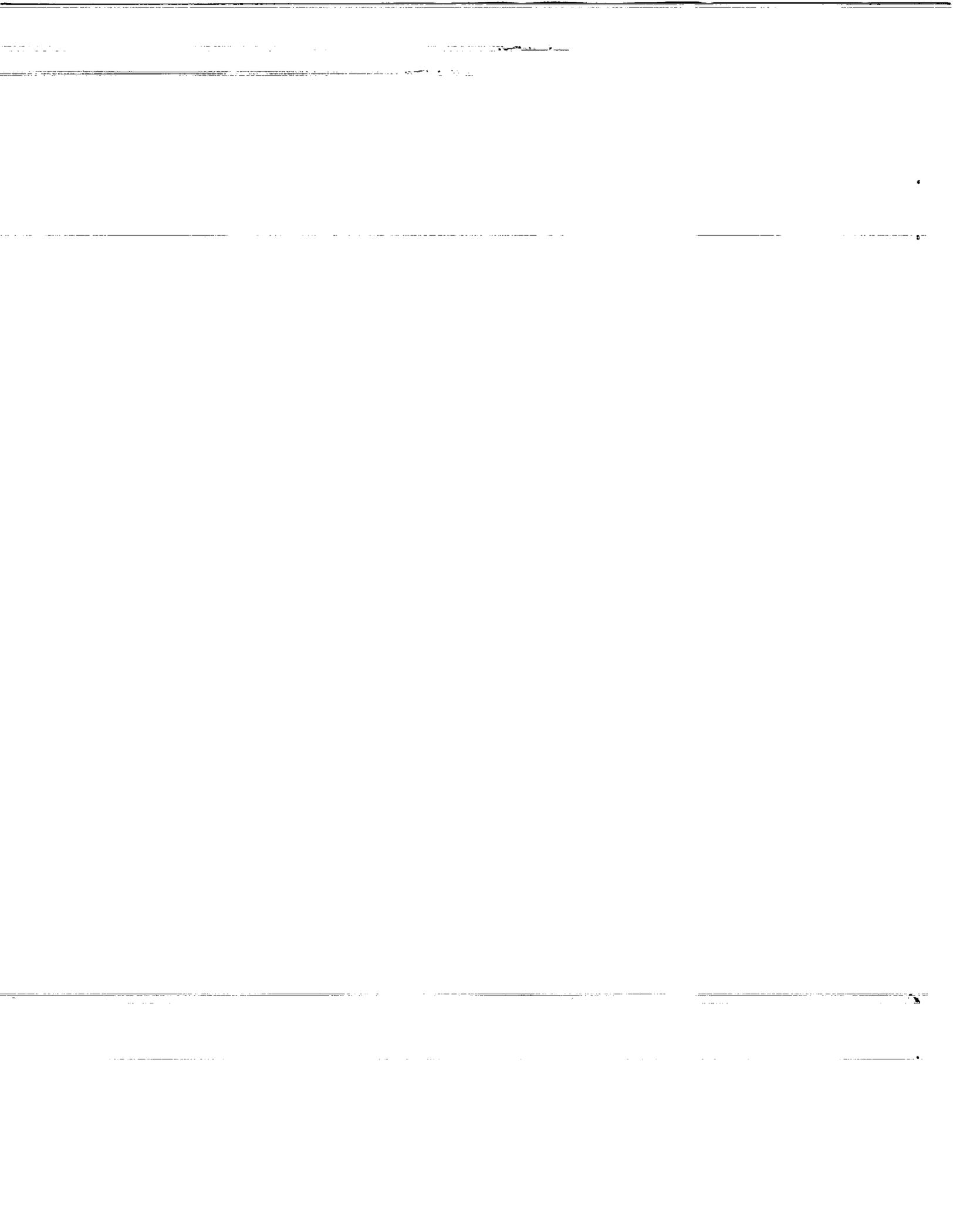
PHILIP LAIRD

AI RESEARCH BRANCH, MAIL STOP 244-17  
NASA AMES RESEARCH CENTER  
MOFFETT FIELD, CA 94035

**NASA** Ames Research Center  
Artificial Intelligence Research Branch

Technical Report RIA-89-01-07-0

January, 1989



# A Survey of Computational Learning Theory

Philip D. Laird  
Artificial Intelligence Research Branch  
NASA Ames Research Center  
Moffett Field, California 94035

## 1 Overview

I write this survey mainly for readers interested in machine learning, whose research methodology includes some amount of mathematical modeling. The scope of the article is limited to research that has a formal foundation and pertains to the problem of making computers “learn”, whatever that means. I hope that readers new to the field find a roadmap to the recent literature, and that researchers already immersed in the field may gain some perspective about the relationship of their work to that of their colleagues.

Like most surveys, this article contains no new results, except perhaps for relating research that previously had not been compared. Most of the work is generally available in the published literature, or at least in technical research reports still available from the sponsoring institutions. The remarks in this introduction are my own thoughts and are not intended to express any consensus on the part of learning researchers.

**The Nature of Learning.** In thinking about the content and structure of this survey, I have once again been forced to confront the annoying question: what is learning? Many programs and models purport to learn in some fashion, but like intelligence, learning is easier to recognize than to define. The “Potter Stewart” mode of concept definition (“I know it when I see it”) is of no help when an author is forced to declare his prejudices, as I am now.

What is the nature of a program that learns? Most programs  $P$  compute a relation: given  $x$ , output a value or set of values  $y = P(x)$  with a specified property. We assume that the initial state of the program is always the same at the beginning of each computation, so that regardless of any previous computations, the output values  $P(x)$  depend only on  $x$ . A *learning* program, however, also modifies its initial state: starting in state  $q$ , the result of the computation  $P(x | q)$  is a sequence of values  $y$  and a new state  $q'$  from which subsequent computations will begin. It is this progression of states of the program that interests us, that is the result of learning. For cumulative changes of state to be considered learning, there must also be improvement in the computation.

types of problems?

In a field where the final concern is with what tasks can be accomplished by computer, one often hears complaints that formalism offers little or no insight. Such comments may reflect a misunderstanding of the term "formal"; in particular "formalism" and "mathematics" are commonly taken to be synonymous. But a well-designed machine or program requires the same insight, clarity of reasoning, and careful structure as the best mathematical work. "Formal" here refers to the careful expression of ideas in a manner suitable for succinct communication. The ideas we are most interested in are those with the broadest implications, whether for designing programs or reasoning about concepts.

Mathematical formalism can serve machine learning in several ways. By identifying fundamental concepts common to many learning problems, the theory reduces the number of ideas we must retain in order to solve problems. By modeling and simplifying problems so that we can reason formally about them and their solutions, we increase our understanding of the relevant elements of the problem domain and how they affect the solution. In the analysis of learning algorithms we discover their capabilities and limitations. Occasionally a negative result will show that acceptable solutions to the problem do not exist as specified, and thereby force us to revise our approach to the problem.

Not all formal work, however, serves us equally well in the task of enabling machines to learn. As new models proliferate, and algorithms for learning with respect to those models are published, people question the relevance of the research. When a theorist is asked, "Have you implemented any of your results?" one sometimes hears in response: "My results have been proved formally and rigorously; therefore, programming them is just an exercise that will not contribute any new ideas." Such comments demonstrate a misunderstanding of the role of theory. Mathematics, especially as a model of learning, is *just* a model. Models incorporate a host of assumptions and simplifications in order allow mathematical treatment. As a consequence, the theorems are merely suggestive of the true phenomenon being studied; only experience with implementation can determine whether the resulting concepts are useful. When the theory arises in response to experience – perceptron theory, for example – the mapping from theory to practice is usually fairly close. But since much of the learning theory to date has been concerned less with solving concrete problems than in formulating abstract models of learning, its relevance to machine learning remains to be demonstrated. The reason for this situation is not hard to find: under pressure to get results, theorists in all areas occasionally pursue a mathematically fertile theory beyond the limits of its usefulness.

Both mathematical and machine studies of learning are vulnerable to the "sirens of detail". Some theorists, for example, develop their ideas in increasingly abstract terms, with few of their colleagues able to understand, let alone utilize, the products of their research. Likewise there are those who derive pleasure from writing an enormously complex machine program, perhaps to emulate certain aspects of human intelligence or to display characteristics that even Potter Stewart would recognize as learning. In each case these "sirens" may be luring the researcher away from the true objective of the formal science of learning, which is to discover its conceptual kernel and to make these concepts part of the common knowledge of the community of computer

scientists.

For such a theory to become common knowledge in the way that the periodic table and Newtonian mechanics are common knowledge, *there must exist simple, unified bases for the variety of behaviors collectively called learning*. No one knows whether such a theory exists. But we hope that it does, and we continue to direct our research efforts in its pursuit. One of the goals of this survey is to highlight the elements common to different learning models.

**General Outline.** This article is in four parts. The first two sections view learning as an inductive, rather than a deductive, inference process. The first section treats induction in first-order logic, and the second does so in probabilistic logic. After that, we review research that treats learning as a combinatorial search problem in which the choice of representation is crucial to the time complexity of the problem. Finally we examine learning in networks of simple processing units that communicate only with other units nearby.

In each section, I describe the principal ideas, and some of the fundamental results when these can be stated without excessive exposition. I follow with a description of the representative literature so that the reader can begin to locate source materials in the area. I have not compiled a complete bibliography; nor is it possible in these few pages to do justice even to the results that are described.

The wealth of research results in learning has made the decision about what to include extremely difficult. No reader should presume that all the interesting and important work in formal learning theory has been listed here; equally, no researcher should attribute my omission of his or her results as a judgment of worth. I have included research results that, in my view, assist the reader to acquire a coherent picture of the field, understand the motivation for current research, and appraise its significance.

## 2 Induction

From the earliest days of AI, researchers have pursued two main approaches to the automation of intelligence:

- explicit knowledge representation, coupled with general inference algorithms for utilizing the consequences of that knowledge;
- “self-organizing systems” composed of massive assemblies of small-scale units, wherein knowledge is distributed throughout the system, and intelligence emerges as a macroscopic property of the system as a whole.

Mathematical logic forms the theoretical basis for the first approach. By contrast, no single branch of mathematics dominates the formal theory in the second, although thermodynamics and statistical mechanics have provided some of the inspiration.

In this section and the next we consider learning within the first paradigm, that of inference-based systems. The term “inference” refers generally to effective symbolic

procedures for deriving new facts from known ones. That process is called *deductive* if it fits the schema of a process of logical deduction (axioms, rules of inference, etc.). Much of the philosophical research in learning has been devoted to developing a corresponding theory of *inductive* inference, a calculus for inferring generalizations from particular observations. Whereas theorems in a deductive calculus are subsumed by the axioms, in an inductive system the "facts" or examples are subsumed by the inferences.

Within the framework of the Church-Turing model of computation, a vigorous study of the absolute limits of inductive inference has been conducted for the past twenty years. The result is a rich and beautiful theory of the abstract complexity of learning. But this line of mathematics says little about the design of learning programs for real machines, and in accordance with the stated objectives of this survey, I shall not describe it further. See [62] for an introduction to this research.

**Inductive Synthesis of Concepts.** *Concept learning* has been one of the core problems of machine-learning research, for both theory and practice.<sup>1</sup> In its simplest form, a *concept* is a subset of some universe  $U$  of objects. Associated with the concept is a representation expressed in some chosen language  $\mathcal{L}$ . For example, "dog" denotes a particular subset of the set of all animals, and we can represent this subset by a logical relation over certain features (number of legs, has a tail, sensitivity of the olfactory sense, etc.). In a concept-learning task, the learner receives *examples* labeled "dog" or "not a dog" by a process we call the *teacher* already in possession of the target concept. Based on the examples, the learner offers a sequence of hypotheses, expressed as sentences in  $\mathcal{L}$ . This sequence should eventually converge to the target concept or to a close approximation. This informal characterization of concept learning can be formalized in many ways, depending on how the examples are represented, how the teacher selects the examples, what computational limitations apply to the learner, and so on.

Assuming the representation language  $\mathcal{L}$  is recursively enumerable, there is a very simple procedure (called *identification by enumeration*) for finding the target concept from the examples. Let  $L_1, L_2, \dots$  be a list of the concepts. Suppose  $x_1, x_2, \dots$  is the list of examples presented by the teacher. After receiving the  $i$ 'th example, the inference process chooses as its next hypothesis the first concept in the list that agrees with the  $i$ 'th example. In response, the teacher chooses a counterexample (provided one exists). It is easy to see that, under very general conditions, this simple procedure *converges* — i.e., after some finite number of examples, the process outputs the correct concept. Moreover it enjoys two of the characteristics of a learning algorithm as set forth in the introduction: search and slow storage growth. The third requirement — gradual improvement in the hypotheses — is not fulfilled in this case, and for this reason identification by enumeration does not inspire much excitement as a learning algorithm.

It is equally apparent that, without considerable amplification, identification by

---

<sup>1</sup>Throughout this survey I use the problem of supervised concept learning to illustrate the main ideas and to facilitate comparison of the different approaches. Other learning tasks are treated in the references.

enumeration is not a practical inference procedure. Nevertheless it forms the basis for many learning algorithms, including the Inductive Synthesis method, to be described below. In any real problem a correct hypothesis is liable to be a large and complex expression; consequently any practical algorithm must have a technique for inferring the individual parts of the concept instead of searching sequentially through the entire concept space. Many clever techniques for accomplishing this had been devised for particular languages (automata, grammars, Turing machines, lisp functions, ...). Then Ehud Shapiro devised an elegant way to do inductive synthesis using first-order logic as a representation language. As a result we can now solve most of these other concept-learning problems in a unified way by representing the hypotheses in a first-order language.

I shall give an overview of Shapiro's theory, based on [76]. For this discussion, we represent concepts as first-order Horn sentences — that is, conjunctions of zero or more clauses of the form  $(P_1 \wedge \dots \wedge P_k) \rightarrow Q$ , where  $P_i$  and  $Q$  are atoms (neither negated nor quantified). Each clause is universally quantified over its variables; existential quantification is not used. The  $P_i$ 's are called the *premises* and  $Q$  the *conclusion* of the clause. For this discussion, the term "clause" means "Horn clause". **T** and **F** denote "true" and "false", respectively. As logical inference rules we allow resolution and substitution for equals.

For example, the following sentence, consisting of two clauses, defines the concept of *plus* as the set of triples  $(x, y, z)$  such that  $x + y = z$ . Zero (0) is a constant symbol in the language, and the intended interpretation of the function symbol *succ* is that *succ*( $x$ ) means  $x + 1$  (successor function).

$$\begin{aligned} \mathbf{T} &\rightarrow \text{plus}(X_1, 0, X_1). \\ \text{plus}(X_2, Y_2, Z_2) &\rightarrow \text{plus}(X_2, \text{succ}(Y_2), \text{succ}(Z_2)). \end{aligned}$$

The first clause expresses the fact that  $x + 0 = x$  for all (natural integers)  $x$ ; the second says that  $x + (y + 1) = (x + y) + 1$ . Together these define inductively the concept *plus* over all terms of the form 0, *succ*(0), *succ*(*succ*(0)), etc. Note that the two clauses are implicitly quantified by  $\forall X_1$ ,  $\forall X_2$ ,  $\forall Y_2$ , and  $\forall Z_2$  and conjoined together.

Let  $\mathcal{L}$  be a fixed first-order language with equality and at least one constant symbol.  $U_{\mathcal{L}}$  denotes the Herbrand universe of variable-free terms over  $\mathcal{L}$ . In the example above,  $U_{\mathcal{L}}$  is the set  $\{0, \text{succ}(0), \text{succ}(\text{succ}(0)), \dots\}$ . Let  $p$  be an  $n$ -place predicate in  $\mathcal{L}$ ; a *model*  $M(p)$  of  $p$  is a set of atoms  $p(t_1, \dots, t_n)$ , where each  $t_i$  is a term in  $U_{\mathcal{L}}$ . Thus the model associated with the predicate symbol  $p$  is a concept — namely, a subset of  $U_{\mathcal{L}}$ . A *model*  $M$  over  $\mathcal{L}$  is the union of models  $M(p)$  over all predicate symbols  $p$  in  $\mathcal{L}$ .

The objective of the Inductive Synthesis system is to find a sentence  $\varphi$  in  $\mathcal{L}$  representing the model  $M$  in the following way: the set of variable-free atoms, or *facts*, that are logical consequences of  $\varphi$  is precisely the set  $M$ .

For example, the predicate *plus* defined above is one representation for the model consisting of the set of facts:

$$\text{plus}(0, 0, 0)$$

$\text{plus}(0, \text{succ}(0), \text{succ}(0))$   
 $\text{plus}(\text{succ}(0), 0, \text{succ}(0))$   
 $\text{plus}(\text{succ}(0), \text{succ}(0), \text{succ}(\text{succ}(0)))$   
 ...

This same model can also be represented in many other ways.

We assume that there is a teacher who knows what  $M$  is and who can provide information about  $M$  in various ways. Among these is to give a counterexample to a hypothesis  $\varphi$ , if  $\varphi$  does not exactly represent  $M$ . For example, suppose our hypothesis for the plus concept above is

$$\mathbf{T} \rightarrow \text{plus}(X, Y, 0)$$

(i.e., for all  $x$  and  $y$ ,  $x + y = 0$ ). Then a counterexample to this hypothesis would be

$$-\text{plus}(0, \text{succ}(0), 0).$$

The flag “-” in the example signals that this atom is not in the target model  $M(\text{plus})$ . Again, in response to the hypothesis

$$\mathbf{T} \rightarrow \text{plus}(X, \text{succ}(Y), Y).$$

the teacher might return, as counterexamples, either of the following:

$$+\text{plus}(0, 0, 0)$$

$$-\text{plus}(0, \text{succ}(0), 0)$$

Note that we are expecting the teacher to have “superpowers”, since in general no recursive computation exists that can answer correctly all such queries. However for many expressive domains of practical concern, these are either decidable questions or approximately decidable in the sense that an algorithm can provide the answers with high probability, or, at worst, answer, “I don’t know”.

The Inductive Synthesis algorithm is a variant of the idea of identification by enumeration, with an important change: when the hypothesis is found to be incorrect, the unit of modification is the clause, not the entire hypothesis. That is, when given a counterexample for its current hypothesis, the algorithm either discovers an incorrect clause and modifies it, or supplies an additional clause needed to account for additional positive examples.

A simplified version of the algorithm is as follows:

#### Inductive Synthesis Algorithm (Outline)

1. Initialize the current hypothesis  $\varphi$  to the empty sentence. (The empty sentence has no clauses and thus represents the null-set model.)
2. While  $\varphi$  is incorrect, repeat the following.
  - 2.1 Obtain from the teacher a counterexample  $\pm e$  and store it.

2.2 Repeat the following until  $\varphi$  is correct for all stored facts.

2.21 If  $\varphi$  fails to imply some stored *positive* example, find a predicate  $p \in \mathcal{L}$  such that, for some fact  $e \in M(p)$ ,  $e$  is not a consequence of  $\varphi$ . Then add to  $\varphi$  a clause that covers (implies) the fact  $e$ .

2.22 If  $\varphi$  implies some stored *negative* example, find an incorrect clause in  $\varphi$  and remove it.

3. Write down  $\varphi$  as the solution.

As a slightly more elaborate example, consider the target sentence consisting of the two previous clauses defining *plus*, together with the following two clauses defining *times*:

$$\mathbf{T} \rightarrow \mathbf{times}(X_3, 0, 0).$$
$$\mathbf{times}(X_4, Y_4, W_4) \wedge \mathbf{plus}(X_4, W_4, Z_4) \rightarrow \mathbf{times}(X_4, \mathbf{succ}(Y_4), Z_4).$$

We start the algorithm with the empty sentence as the current hypothesis, expressing the conjecture that no term is the sum of any other terms nor the product of any other terms. In response, assume the teacher returns the counterexample,

$$+\mathbf{plus}(\mathbf{succ}(0), \mathbf{succ}(0), \mathbf{succ}(\mathbf{succ}(0))).$$

The algorithm, noting that this fact is not covered by the empty sentence, determines that the theory of the predicate *plus* is incomplete and adds a clause, say

$$\mathbf{T} \rightarrow \mathbf{plus}(X, Y, Z),$$

to  $\varphi$ . Next, the teacher tells us this hypothesis is too general by providing the counterexample  $-\mathbf{plus}(0, 0, \mathbf{succ}(\mathbf{succ}(0)))$ . In response the algorithm removes the clause, but is now back where it started since  $\varphi$  no longer covers the first example. So it searches again, this time for a less general clause that both implies the first example and fails to imply the second. For example, the clause

$$\mathbf{plus}(X, Y, \mathbf{succ}(X))$$

will cover these two examples.

Either this pattern of hypothesis-and-counterexample repeats forever, or the algorithm converges to a sentence equivalent to the target definitions of *plus* and *times*, at which point the teacher cannot return any counterexample and the algorithm halts with a correct sentence.

To complete the description of the algorithm, we also need

- a diagnosis procedure for finding an incomplete predicate or erroneous clause causing the failure;
- a search procedure to find an appropriate clause to cover a missing fact.

These I shall describe presently. Another requirement is that the algorithm be able to decide whether its hypothesis  $\varphi$  implies a given atom  $e$ . In general this problem is partially undecidable, but as a practical matter, useful concept representations often come with a bound on the time required to decide membership in that concept. If the algorithm cannot prove or disprove membership of an example within the bounded number of steps, it may conclude that the hypothesis fails on that example.<sup>2</sup>

**Diagnosis procedure.** The diagnosis problem is typical of the “credit assignment” dilemmas that often arise in search problems. Continuing the above example, suppose the algorithm has reached the point where the hypothesis  $\varphi$  has the correct definition of **times**, but has not yet acquired the full definition of **plus**. The example

$$+\text{times}(0, \text{succ}(\text{succ}(0)), 0)$$

will be a counterexample to  $\varphi$  if, for example, only the clause

$$\mathbf{T} \rightarrow \text{plus}(X, 0, X)$$

is missing from  $\varphi$ . The algorithm must somehow determine that it is the **plus** predicate that is in error, not **times**.

That the diagnosis problem is solvable depends on the following two lemmas:

- If  $\varphi$  implies a fact not in  $M$ , then  $\varphi$  contains an incorrect clause — more specifically,  $\varphi$  contains a clause  $C$  not valid in  $M$ .
- If  $\varphi$  fails to imply a fact in  $M$ , then  $\varphi$  is incomplete — more specifically, there exists a clause  $C$  valid in  $M$  but not implied by  $\varphi$ .

These are fairly obvious when the target concept has only one predicate (as in the **plus** example above), but with more than one, the interactions among the various clauses make it difficult to assign blame within  $\varphi$ . These simple lemmas are crucial in telling us that the problem can be traced to at least one predicate  $p$  and its target  $M(p)$ .

To diagnose a negative counterexample, proceed as follows. We assume the teacher can answer *membership queries* of the form: “Is the ground atom  $p(\dots)$  in  $M(p)$ ?”. First, construct the resolution proof  $\varphi \vdash e$  (where  $e$  is a negative example). Let  $(P_1 \wedge \dots \wedge P_n) \rightarrow Q$  be the clause in which  $Q$  unifies with  $e$  via the substitution  $\theta$ . For each of the premises  $P_i$  we ask the teacher if the fact  $\theta(P_i)$  is in  $M$ . If the response is yes for all  $i$ , then this clause is erroneous, since true premises are implying a false result; diagnosis then returns this clause. Otherwise, let  $\theta(P_i)$  be false in  $M$ . Since  $\varphi \vdash \theta(P_i)$ , we recursively diagnose this false example.

To diagnose a positive counterexample, proceed as follows. We assume the teacher can answer *existential queries*: given an atom  $p(t_1, \dots, t_n)$  in which some of the terms  $t_i$  contain variables, enumerate the instantiations of the variables that yield atoms in  $M(p)$ . Let  $e$  be a positive example not covered by  $\varphi$ . Since  $\varphi \not\vdash e$ , one of two situations

---

<sup>2</sup>This formalises a heuristic we use to decide whether our programs are looping endlessly or not.

must hold: (1) no clause in  $\varphi$  unifies with  $e$  — in which case the predicate  $p$  in the atom  $e$  is incompletely covered by  $\varphi$ , and the diagnosis returns  $p$ ; or (2) every clause in  $\varphi$  whose conclusion unifies with  $e$  has at least one premise  $P_i$  not implied by  $\varphi$ . For each such premise, we ask the teacher to enumerate the instances of  $P_i$  that are true in  $M$ , and if  $\varphi$  fails to imply any of these, we recursively investigate these for incompleteness.

**Refinement search.** The search for a new clause to cover a missing fact  $+e$  plays a vital role in the correctness and efficiency of the learning procedure. The search must be complete, in that no possible clause may be overlooked indefinitely as a candidate. The search must also be systematic: once a clause has been discarded it should never again be tried. Shapiro accomplishes this by defining a well-founded partial ordering  $\preceq$  on clauses whose conclusion contains the predicate  $p$ . *Well-founded* means that the ordering should be semi-infinite, with no infinite descending chains. In addition, the relation  $\preceq$  is chosen so that  $C_1 \preceq C_2$  only if  $C_1$  subsumes  $C_2$ . An ordering with these properties is called a *refinement* relation.

We use this ordering as follows:

- Whenever a clause is removed from a hypothesis, that clause is *marked*.
- When searching for a clause to cover a missing fact, we select from the unmarked clauses one that is minimal with respect to the ordering  $\preceq$ .

An example of a refinement on Horn clauses over the predicate *plus* is the following:

- The clause that is minimum with respect to  $\preceq$  is  $T \rightarrow \text{plus}(X_1, Y_1, Z_1)$ .
- Given the clause  $C = (P_1 \wedge \dots \wedge P_n) \rightarrow \text{plus}(t_1, t_2, t_3)$ , we form the set of clauses directly beyond  $C$  in the ordering by applying one of the following modifications to  $C$ :

- Unify two distinct variables in  $C$ . For example, from the clause

$$T \rightarrow \text{plus}(X_1, Y_1, Z_1)$$

we obtain

$$T \rightarrow \text{plus}(X_1, X_1, Z_1)$$

and

$$T \rightarrow \text{plus}(X_1, Y_1, Y_1).$$

- Replace all occurrences of a variable  $X \in C$  by a constant or a function in its most general instantiation. For example, from

$$T \rightarrow \text{plus}(X_1, Y_1, Y_1)$$

we can obtain, among other clauses,

$$T \rightarrow \text{plus}(X_1, 0, 0)$$

and

$$T \rightarrow \text{plus}(\text{succ}(X_1), Y_1, Y_1).$$

- o Add as a new premise to the clause  $C$  an atom in its most general instantiation. For example, from

$$T \rightarrow \text{plus}(\text{succ}(X_1), Y_1, Z_1)$$

we can obtain, among other clauses,

$$\text{plus}(X_2, Y_2, Z_2) \wedge T \rightarrow \text{plus}(\text{succ}(X_1), Y_1, Z_1)$$

and

$$\text{times}(X_2, Y_2, Z_2) \wedge T \rightarrow \text{plus}(\text{succ}(X_1), Y_1, Z_1).$$

With this refinement, we will eventually generate every possible clause (or an equivalent to every possible clause), and are thereby assured that search for a clause to satisfy a set of examples will terminate successfully.

**Convergence properties.** We have now sketched the main features of the Inductive Synthesis algorithm: the basic algorithm itself, the requirements of the teacher, the diagnosis algorithm, and the search-for-clause mechanism using a refinement relation. The main theoretical result about this algorithm is that it works: *For any model  $M$  over  $\mathcal{L}$  that can be expressed with a Horn sentence, the algorithm converges in a finite number of iterations to a Horn sentence  $\varphi$  such that, for any fact  $f$ ,  $\varphi \vdash f$  iff  $f \in M$ .*

The way the algorithm obtains just the information it needs from the teacher, locates errors within the current hypothesis, and searches systematically for the right combination of clauses are all highly original contributions to the theory and practice of inductive inference. Its limitations are that it stores all examples (violating the low-storage desideratum), relies on totally accurate information from the teacher, and requires explicit examples of all predicates  $p \in \mathcal{L}$ , not just examples of the target concept. For example, we might want the algorithm to see examples of the predicate `times` and infer that some intermediate predicate (`plus`) is required in the definition of `times`. The inference of auxiliary concepts is recognized as a difficult problem, for which no one has yet found a satisfactory solution.

**Sources.** Identification in the limit was first defined and studied by Gold [33]. An excellent survey of inductive inference theory and techniques is to be found in the review article by Angluin and Smith [7]. The Inductive Synthesis algorithm began as the Model Inference System of Ehud Shapiro [78, 77], and was eventually recast as a system for synthesis of Prolog programs from examples of their input/output behavior [76]; the latter monograph also contains a detailed implementation.

The elegance of Shapiro's algorithm has inspired other authors to explore ways of generalizing and applying the ideas. The algebraic (as opposed to logical) basis of the algorithm was exposed by Laird [51]. Angluin [13] compares the power and complexity of several types of query capabilities in teachers, including the ones employed by Shapiro. Whereas Shapiro's refinement search replaces a clause by one that is more specific, others [39, 24] have employed generalization in their search, replacing a clause by one that covers more examples. Algorithms that are constructive (formulate

hypotheses directly from the examples) rather than enumerative (like Shapiro's algorithm) are potentially more efficient; a few such algorithms have been found, including a subclass of logic programs [72] and linear grammars [82].

### 3 Bayesian Induction

As a representation language for AI, first-order logic leaves unresolved many difficulties, among them how to incorporate plausible deductions from uncertain premises in a formally consistent way. Indeed, the entire subject of uncertainty reasoning is an active research topic in AI. Curiously, the basis for much of the current research on plausible reasoning was discovered more than two centuries ago: Thomas Bayes's famous theorem on conditional probabilities has enabled scientists from Laplace and Gibbs to Keynes and Shannon to derive statistically sound inferences from noisy data.

In this section we continue with the general problem of how to make inferences by induction; but instead of first-order logic, we shall adopt *probabilistic logic*, with emphasis on the learning aspects. The general situation is familiar: we have some sample data, and we have a choice of hypotheses with which to explain the data. But now, because of noise and other random factors, the data can serve only as evidence for and against certain hypotheses, not as counterexamples to eliminate incorrect hypotheses. How, then, do we choose a hypothesis using the evidence? And as new data arrive, how can we revise our hypotheses so that eventually we converge? And finally, given that we converge, do we necessarily converge to a good hypothesis?

**A Calculus of Beliefs.** A formal logic of plausible deduction extends Boolean logic by capturing formally the sorts of deductions that humans perform every day:

- if  $A$  frequently implies  $B$  and  $A$  occurs, then  $B$  is more plausible;
- if  $A$  implies  $B$  and  $B$  occurs, then  $A$  is more plausible;
- if  $A$  implies  $B$  and  $A$  is known to be false, then  $B$  becomes less plausible.

An essential requirement of such a logic is that it be *consistent*: from the same evidence, all paths of reasoning should produce the same conclusions. The logic should also subsume standard Boolean logic when all probabilities are 1 or 0.

In place of implications, we introduce *conditionals*. The conditional  $(A | B)$  represents, roughly, the possibility that proposition  $A$  holds, given the certain knowledge that proposition  $B$  holds. Similarly the conditional  $(A \wedge B | C \wedge D)$  represents the possibility that both  $A$  and  $B$  hold, given the certainty of both  $C$  and  $D$ . Probabilistic logic replaces the validity of propositions by a measure of belief, a function ("probability function") that assigns a value to every conditional.

The principal problem of probabilistic logic is to determine the requisite properties of every appropriate measure of belief. Formally this is the reverse of mathematical probability theory, where a probability measure is defined axiomatically, and theorems are obtained relating the measures of different sets. To serve as a useful model of a

rational agent, a probability function should satisfy certain *desiderata*. For example, as the probability  $p(A | B)$  increases, then  $p(\neg A | B)$  should decrease. Again, if  $(A' | C)$  is more probable than  $(A | C)$ , then for any event  $B$ ,  $(A' \wedge B | C)$  should be at least as probable as  $(A \wedge B | C)$ . Standard probability theory (assigning real numbers to measurable sets) satisfies these and other desiderata, and by general agreement fulfills the requirements of a calculus of beliefs. But is it the *only* such way to compute beliefs?

There have been many attacks on this question, starting with Keynes in 1921. Cox, in 1946, proposed a set of desiderata for a "rational" agent and asked whether all logics satisfying the same desiderata are isomorphic to probability theory. In effect his answer was "yes": if beliefs are real numbers, and if events are formulas in propositional logic, and we take as axioms the product rule<sup>3</sup>

$$p(A \wedge B | C) = p(A | B \wedge C)p(B | C)$$

and the sum rule

$$p(A | B) + p(\neg A | B) = 1$$

(where  $1 \equiv p(B | B)$ ), then any consistent logic coincides with probability theory. But, as others have observed, humans carry out plausible inference quite well without real numbers — e.g., with just the values of "likely", "unlikely", "certain", and "impossible". Evidently the axioms proposed by Cox are too strong in their insistence that "probabilities" be real numbers.

Cox's work has since been generalized, most recently by Aleliunas [2]. An example of a set of axioms weak enough to construct a consistent belief logic based on non-numerical probabilities is shown in Figure 1. The set  $\mathcal{L}$  of formulas is the collection of expressions over a finitely generated Boolean algebra, with propositions  $\{A, B, \dots, T, F\}$ , maximum ( $T$ ) and operations  $\wedge, \vee, \neg$ . Conditionals are elements of  $\mathcal{L} \times \mathcal{L}$ . Probabilities are elements of a partially ordered set  $(\mathbf{P}, \leq)$ . For example,  $\mathbf{P}$  could be the reals under the usual total ordering, or the set  $\{\text{impossible}, \text{possible}, \text{probable}, \text{certain}\}^+$  with a partial ordering that extends the basic ordering  $\text{impossible} \leq \text{possible} \leq \text{probable} \leq \text{certain}$  to this set. Probabilities are associated with formulas by a function  $p : \mathcal{L} \times \mathcal{L} \rightarrow \mathbf{P}$ , whose properties are governed by the axioms.

The axioms define the necessary properties of the family  $\mathcal{P}$  of all such belief measures. Among the properties of  $\mathcal{P}$  is that probabilities of complementary conditionals  $p(\neg A | B)$  and  $p(A | B)$  are related by a monotone non-increasing function  $c$  on  $\mathbf{P}$ . In the case of ordinary real probability,  $c(x)$  is the function  $1 - x$ . Another property is the existence of an order-preserving dyadic function  $h$ , which in ordinary probability is multiplication. Many familiar properties of ordinary probability theory follow as theorems from the axioms, including the existence and uniqueness of probabilities  $1 \equiv p(A | A)$  and  $0 \equiv p(\neg A | A)$ . But, in contrast to the Keynes/Cox formulation, these axioms admit models that are quite different from standard probability theory.

<sup>3</sup>Cox [27] argues semi-formally from desiderata that the product rule is the only axiom for computing  $p(A \wedge B | C)$  from other conditionals, consistent with our common sense about causality and plausible reasoning. Similarly he argues that the sum rule is necessary, based primarily on the need to be consistent with the product rule.

A theorem of Aleliunas based on these axioms puts the Keynes/Cox results into perspective:

*Suppose  $P$  is totally ordered. For  $p \in P$  and any integer  $n > 0$ , let  $p^n$  denote the value  $h(p, p^{n-1})$ , with  $p^1 = p$ . Suppose also that the set  $P$  of probability values satisfies the following property: for any probabilities  $p \neq 1$  and  $q \neq 0$ , there exists a positive integer  $n$  such that  $p^n < q$ . Then the structure of  $(P, \leq)$  is isomorphic to a subalgebra of real probabilities, with  $h(x, y)$  corresponding to multiplication  $x \times y$  and  $c(x)$  to  $1 - x$ .*

**Bayesian Learning Algorithms.** Having chosen a belief function  $p$  (which, for clarity, we now assume is standard real probability), we can then take up the learning problem. In the Bayesian approach, the learning problem changes in three important ways.

- Whereas for Boolean logic every hypothesis is equivalent to a characteristic function on the set  $U$  with value 1 for points that are in the concept and 0 for points not in the concept, in the Bayesian framework a point is in the concept with a certain probability. Thus a concept hypothesis is equivalent to a generalized characteristic function that assigns  $p(x) \in [0, 1]$  to each point  $x$  in  $U$ .
- Whereas inductive synthesis has at each stage of the learning procedure a hypothesis that represents its current understanding of the target concept, the Bayesian approach does not single out any hypothesis as the current favorite. Instead, at each stage of a Bayesian learning procedure the algorithm assigns a probability to each hypothesis, representing its belief in that hypothesis as the target.<sup>4</sup>
- Whereas the Inductive Synthesis model assumes that a teacher provides counterexamples to any incorrect hypotheses, the notion of "counterexample" is meaningless in the probabilistic setting. Instead a teacher selects points from  $U$  according to some arbitrary process (unknown to the learner), and probabilistically classifies those examples as "in" or "out" according to some target hypothesis  $H$ .

Formally, we assume a set  $U$  of sample points and a set  $\mathcal{H} = \{H_1, H_2, \dots\}$  of hypotheses. The countability assumption for  $\mathcal{H}$  is a matter of convenience, not necessity; but we do require that no two hypotheses in  $\mathcal{H}$  be equivalent. Each hypothesis  $H_i$  assigns to each point  $x$  in  $U$  a probability, which we interpret as the probability that  $x$  is a member of the concept  $H_i$ . For concept learning this probability depends only on the hypothesis  $H_i$  and the point  $x$ ; it is independent of any other events.<sup>5</sup> An example  $e$  is a point  $x \in U$  together with a flag (+ or -). As usual, a "+" flag indicates that

---

<sup>4</sup>From a slight change of viewpoint, the "current hypothesis" in the Bayesian framework can be taken to be the function assigning a belief to each rule. However this view has not been used.

<sup>5</sup>Bayesian learning can also be applied to learning problems other than concept learning. In these situations temporal dependencies may become part of the hypotheses and the way that the sample is presented.

Aleliunas's axioms for a family  $\mathcal{P}$  of belief functions over  $(\mathbf{P}, \leq)$  and  $\mathcal{L}$ :

1.  $\forall p \in \mathcal{P}, A_1, B_1, A_2, B_2 \in \mathcal{L}$ : if  $A_1 \equiv B_1$  and  $A_2 \equiv B_2$  then  $p(A_1 | A_2) = p(B_1 | B_2)$ .
2.  $\forall p \in \mathcal{P}, A \in \mathcal{L}$ :  $p(A | \mathbf{F}) = p(A | A)$ .
3.  $\forall p \in \mathcal{P}, A, B \in \mathcal{L}$ :  $p(A \wedge B | B) = p(A | B) \leq p(B | B)$ .
4.  $\forall p_1, p_2 \in \mathcal{P}, A \in \mathcal{L}$ :  $p_1(A | A) = p_2(A | A)$ .
5. There exists a monotone non-increasing function  $c : \mathbf{P} \rightarrow \mathbf{P}$  such that  $\forall p \in \mathcal{P}, A, B \in \mathcal{L}$ :  $c(p(A | B)) = p(\neg A | B)$ , provided  $B$  is not logically equivalent to  $\mathbf{F}$ .
6. There exists a function  $h : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$  such that  $\forall p \in \mathcal{P}, A, B, C \in \mathcal{L}$ :
  - $h$  is order-preserving on each of its arguments;
  - $p(A \wedge B | C) = h(p(A | B \wedge C), p(B | C))$ ;
  - Let  $0_{p,C} \equiv p(\neg A | A)$ . If  $p(A \wedge B | C) = 0_{p,C}$ , then either  $p(A | C) = 0_{p,C}$  or  $p(B | A \wedge C) = 0_{p,C}$ .
7.  $\forall p \in \mathcal{P}, A, B \in \mathcal{L}$ : if  $p(A | B) \leq p(A | \neg B)$ , then  $p(A | B) \leq p(A | \mathbf{T}) \leq p(A | \neg B)$ .
8.  $\forall x, y, z \in \mathbf{P}$ , and distinct  $A, B, C \in \mathcal{L}$ : there exist functions  $p_1, p_2, p_3 \in \mathcal{P}$  (not necessarily distinct) such that
  - $p_1(A | \mathbf{T}) = x$ ,  $p_1(B | A) = y$ , and  $p_1(C | A \wedge B) = z$ .
  - $p_2(A | B) = p_2(A | \neg B) = x$  and  $p_2(B | A) = p_2(B | \neg A) = y$ .
  - $p_3(A | \mathbf{T}) = x$  and  $p_3(A \wedge B | \mathbf{T}) = y$  whenever  $y \leq x$ .

Figure 1: Axioms of a probabilistic logic.

the point  $x$  is in the target concept, and vice versa for “-”. *But in different examples, the same point  $x$  may be flagged with different signs by the teacher*, because (recall)  $x$  is in the concept only with some probability  $p$  and out with probability  $1 - p$ . If the target hypothesis assigns probability  $p$  to  $x$ , then on average the teacher will flag  $x$  “+” a fraction  $p$ , and “-” a fraction  $1 - p$  of the times that  $x$  is presented.

The elementary propositions in  $\mathcal{L}$  are of two types: (1) “ $H_i$  is the target hypothesis”; and (2) “the example  $e$ ”. The conditional  $(+x | H_i)$  is the event that the teacher classifies the point  $x$  as “in” the concept, given that the target hypothesis is  $H_i$ . We ordinarily assume that  $p(+x | H_i)$  can be computed from knowledge of both  $x$  and  $H_i$ . Clearly  $p(-x | H_i) = 1 - p(+x | H_i)$ . The conditional  $(H_i | e)$  is the event that  $H_i$  is the target hypothesis used by the teacher, given the example  $e$ .  $p(H_i | e)$  denotes our belief in  $H_i$ , given  $e$ . To compute this probability is the objective of the learning algorithm.

The basis of the learning algorithm is Bayes’s Theorem. It may appear that we have lost Bayes’s Theorem in the formalism above, but it can be found lurking in Axiom 6 of Figure 1:

$$p(A \wedge B | C) = h(p(A | B \wedge C), p(B | C)).$$

Since  $A \wedge B \equiv B \wedge A$ , we have (by the first axiom)

$$h(p(A | B \wedge C), p(B | C)) = h(p(B | A \wedge C), p(A | C)).$$

When  $h$  is ordinary multiplication, we immediately recognize Bayes’s rule.

I shall first describe the general procedure without reference to Bayes’s rule, and then consider what it all means. The learning algorithm proceeds in “stages” — receiving an example, updating beliefs in each hypothesis, receiving another example, and so forth. By the term “stage  $k$ ” we refer to the interval after examining the  $k$ ’th example  $e_k$  but before examining the  $k + 1$ ’st. The symbol  $e_k$  indicates the sequence  $e_1 e_2 \dots e_k$  of examples examined in the first  $k$  stages. At each stage  $k$  in the learning process, including the initial stage ( $k = 0$ ) before any data have been seen, there is a bias function  $\beta_k: \mathcal{H} \rightarrow [0, 1]$  expressing the learner’s preference for each of the hypotheses. This bias is normalized:<sup>6</sup>

$$\sum_i \beta_k(H_i) = 1. \tag{1}$$

At stage  $k$ , our bias  $\beta_k$  toward any hypothesis  $H_i$  will depend on the observations  $e_k$  as well as our initial bias  $\beta_0$ . A larger value for  $\beta_k(H)$  indicates a stronger belief in the hypothesis  $H$ .

To compute the bias for any hypothesis  $H_i$  given the input sequence  $e_k$ , the algorithm uses the following formula:

$$\beta_k(H_i) = c \beta_0(H_i) p(e_k | H_i). \tag{2}$$

Here,  $c$  is a normalizing constant chosen to satisfy (1). The factor  $p(e_k | H_i)$  is called the *likelihood* of the evidence  $e_k$  given the hypothesis  $H_i$ . Thus we can interpret (2) as

<sup>6</sup>One sometimes allows  $\beta_0$  to be unnormalised, a so-called *improper prior*.

changing the new bias for  $H_i$  in proportion to the prior bias  $\beta_0(H_i)$  and the likelihood that the hypothesis  $H_i$  would have produced the observations. If both factors are large, our belief in the hypothesis is also large. But a large likelihood can be counteracted by a small prior bias, or vice versa. For example, in most locations our prior belief that an earthquake will occur is so small that, when the building shakes, we are much more likely to conclude that a truck has passed nearby, despite knowing that an earthquake would likely produce just such vibrations.

In (2) we have an algorithm for computing the bias function  $\beta_k$  from the example data, but what does this quantity mean? It is natural to interpret  $\beta_k(H_i)$  as "the probability that the correct hypothesis is  $H_i$ ", although this sentence makes no sense in the context of classical probability theory. By letting  $\beta_k(H_i) \equiv p(H_i | e_k)$  in (2), and setting  $1/c = \sum_j p(e_k | H_j) \beta_0(H_j)$ , we again have a formal statement of Bayes' rule for the probability of (belief in) the hypothesis  $H_i$  given the evidence  $e_k$ . The initial bias  $\beta_0$  should reflect our prior beliefs in the hypotheses. (See the discussion of priors, below.)

What is the result of this algorithm after processing a large sample of data? If one of our hypotheses  $H_i$  is correct, and if the teacher selects points in a reasonable way (e.g., selects them at random), then if the learning algorithm is sound we should expect  $\beta_k(H_j)$  to converge uniformly to

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

in the limit as  $k \rightarrow \infty$ . To prove such a property we would need to know more about the domain, the choice of  $\beta_0$ , and the process for selecting examples; but given these, the requisite proof techniques are well known. Moreover, the convergence property usually holds regardless of the choice of prior bias  $\beta_0$ , provided that  $\beta_0(H_*) > 0$  for the target hypothesis  $H_*$ .

For the Bayesian learning algorithm (2) to be a practical learning algorithm, we need to be able to compute  $p(e_k | H_i)$  rather easily. In concept learning we already have a simplifying assumption that makes this possible, namely that the teacher classifies each example independently of the other examples. Thus

$$p(e_k | H_i) = p(e_1 | H_i) \dots p(e_k | H_i).$$

Another useful feature of the Bayesian learning algorithm is the potential for incremental revision of beliefs. As more data are obtained, it may not be necessary to calculate the probability  $\beta_{k+1}(H)$  from scratch using all  $k + 1$  data values. Often we can use our previous result  $\beta_k$  and the new data value  $e_{k+1}$  to obtain  $\beta_{k+1}$ , and thereby free ourselves from having to store a complete history of the data. The idea is based on the following simple calculation:

$$\begin{aligned} \beta_{k+1}(H) &= p(H | e_{k+1}) \\ &= p(H | e_{k+1}, e_k) \\ &= c p(e_{k+1} | e_k, H) p(H | e_k) \\ &= c p(e_{k+1} | e_k, H) \beta_k(H) \\ &= c p(e_{k+1} | H) \beta_k(H). \end{aligned}$$

The last equality is a consequence of the independent-classification assumption. Again,  $c$  is some normalizing constant. Thus  $\beta_{k+1}(H)$  can be computed knowing only  $\beta_k$  and the last example  $e_{k+1}$ . The independence assumption is critical to this argument and should therefore be examined carefully before adopting it in practice.

**Bayesian Inferences** Now let us suppose that we employ a Bayesian learning algorithm in some domain, and that we are confident the algorithm converges in the limit to the correct hypothesis. At stage  $k$  what have we learned by computing  $\beta_k$ ? Convergence in the limit is not particularly interesting unless we can draw useful inferences and make good decisions based on those inferences at finite times as well.

Following is an algorithm for inferring what we expect to observe at time  $k+1$  in the way of concept membership. Let  $x$  be any point in  $U$ ; write  $p(+x_{k+1})$  to indicate the probability that  $x$  will be classified as "+" if selected by the teacher for presentation at stage  $k+1$ . The probability of the example  $+x_{k+1}$  given our current bias  $\beta_k$  is calculated from the formula:

$$p(+x_{k+1}) = \sum_i p(+x_{k+1} | H_i) \beta_k(H_i). \quad (3)$$

Computing  $p(+x_{k+1})$  for one point  $x_{k+1}$  entails computing an average (over all hypotheses) of the likelihoods weighted by the biases. When  $\mathcal{H}$  consists of a large number of hypotheses, this may not be a practical calculation to do exactly, but at least in theory we have the basis for making principled decisions from our predictions (by using a minimax strategy, for instance). Note that instead of choosing one of the hypotheses as our current favorite and making guesses based on that, we are basing predictions on all hypotheses, weighted by our beliefs in them. Thus the predictions may not coincide with those of any individual hypothesis in  $\mathcal{H}$ .

This is *one* algorithm for predicting concept membership, but is it a *correct* algorithm? After all, many algorithms are possible for choosing hypotheses and making inferences. If someone presented one of these other learning/inference algorithms, how might we argue that the Bayesian algorithm is as good or better?

Mathematically, of course, this question is nonsensical. We can point to our probability model and assert that our inference method is consistent with a rational agent based on certain axioms, and that these axioms satisfy certain desiderata specifying how a rational agent should act. But lacking formal criteria for the quality of an algorithm, we cannot prove one to be better than another.

For certain problems, however, useful criteria are available and formal results have been obtained. In many pattern recognition problems, examples are selected randomly by the teacher, and the stated objective is to minimize the mean squared difference between the true probabilities and those predicted by the target model  $H_*$ . Using this criterion, we can characterize an optimal algorithm as follows. For the point  $x \in U$  let  $F(+x)$  be the actual probability that  $x$  is classified as belonging to the concept. ( $F$  might not correspond to any hypothesis in  $\mathcal{H}$ .) Let  $p(+x | e_k)$  be the probability predicted by an algorithm  $A$  based upon the data  $e_k$ . (When  $A$  happens to be the Bayesian algorithm, this is given by (3) above.) Then the *mean squared*

error,  $Err(A | F, e_k)$ , of the algorithm  $A$  based on  $e_k$  is given by

$$Err(A | F, e_k) = \sum_{x \in U} [p(+x | e_k) - F(+x)]^2,$$

and the *net mean squared error*  $Err(A | F)$  is the expectation  $E_k$  of  $Err$  over all sequences  $e_k$  of  $k$  examples:

$$Err(A | F) = E_k \sum_{x \in U} [p(+x | e_k) - F(+x)]^2.$$

To quantify the overall error of  $A$ , we have to specify the distribution of the problems — i.e., the distribution  $p'(F)$  of all possible  $F$ 's. Then the error of the algorithm is:

$$Err(A) = \sum_F p'(F) Err(A | F). \quad (4)$$

We define an algorithm to be *optimal for the distribution  $p'(F)$*  if it minimizes  $Err(A)$ .<sup>7</sup>

For some classes of pattern recognition problems, the following has been proved: *Suppose a teacher selects a concept from the set  $\mathcal{H}$ , choosing  $H_i$  with probability  $p'(H_i)$  ( $1 \leq i \leq |\mathcal{H}|$ ). Then the Bayesian algorithm based on (9), with prior  $\beta_0(H_i) = p'(H_i)$ , is an optimal algorithm for the distribution  $p'$  of problems. Moreover, this fact is independent of the number  $k$  of examples provided to the algorithm. See [88] for more on this kind of analysis.*

**Priors.** A significant point about the preceding theorem is that, to be optimal, the Bayesian algorithm needs to choose the prior  $\beta_0$  to match the actual distribution of problem instances. But suppose there is *only one* problem instance? What, for example, is the probability of the destruction of the universe in the next century?

The Bayesian learning algorithm requires the learner, before seeing any data, to declare his bias for each of the hypotheses in the form of the function  $\beta_0$ . Besides making all such preferences explicit, this may help the learner to incorporate previous learning experiences and to express requirements external to the learning problem, such as a preference for simpler hypotheses over more complex ones. But this is also a source of controversy, for one must decide how to encode all prior information and preferences in the form of a real-valued function. And to be mathematically convincing, we must do so in a principled way.

Since Laplace, scholars have argued about the nature of *priors* (the function  $\beta_0$ ), to the extent that for much of this century the field of statistical inference has been split into two camps: Bayesians, and those who reject Bayesian inference entirely because of the inherent subjectivity of choosing priors. To illustrate the problem, suppose we are trying to estimate the distance of a particular galaxy from Earth. We have, as data, the results of a small number  $k$  of independent astronomical experiments  $e_k = \{e_1, \dots, e_k\}$ , each of which produces a (noisy) estimate for the distance  $d$ . Being practiced Bayesians, we calculate for each experiment  $e_i$  the distribution  $p(e_i | d)$  of

<sup>7</sup>Note that the number  $k$  of example points given as input to the algorithm is fixed. In effect we are comparing how well different algorithms do when given the same data.

observations given the distance  $d$ , and combine these into a distribution  $p(d | e_k)$  of distances using Bayes's rule:

$$p(d | e_k) = c \beta_0(d) p(e_k | d).$$

But what do we choose for  $\beta_0(d)$ , the prior distribution for  $d$ ? If we assume total ignorance, we might choose a uniform "distribution"  $\beta_0(d) = 1$  for  $d > 0$ .<sup>8</sup> But our ignorance is *not* total: we know that  $d < \infty$ ; with even minimal knowledge of astronomy we can easily write down an upper bound  $d_{\max}$  on this distance. Even if we could agree on a fixed value for  $d_{\max}$ , and choose  $\beta_0(d)$  to be uniform over  $[0, d_{\max}]$ , we would have to admit that, if we are ignorant of  $d$ , we are equally ignorant of any function  $f(d)$ , so why shouldn't we choose  $f(d)$  to be uniform over the interval  $[0, f(d_{\max})]$ ? The problem here is one of quantifying ignorance in a consistent way: in different inference problems, each with the same prior information, we should choose the same prior  $\beta_0$ , even if we are inferring probabilities for different quantities.

No single satisfactory solution to this problem has been proposed, but a number of good ones have been suggested and applied successfully, particularly in domains of scientific inference. One is to select the prior distribution  $\beta_0$  with maximum entropy  $S(\beta_0) = -\sum_i \beta_0(H_i) \log \beta_0(H_i)$  satisfying all "testable" prior information. (A *testable* property of a distribution is one for which an effective decision procedure exists) Another is to construct priors so that the information is invariant under changes in scale (units of measurement) and translation of the coordinate system.

But just as evaluating an inference procedure depends on the definition of an optimal algorithm, any technique for choosing priors can be judged only by how successfully we derive inferences from them, and on no other basis. In all learning problems, we attack the problem by making prior assumptions about the nature of what we are learning; as a minimum this takes the form of choosing a representation for our hypotheses. The success of the learning algorithm depends strongly on the validity of these assumptions; and poor choices show up in the form of answers that predict and explain poorly. Bayesian priors are just another of these initial assumptions. A poor choice of priors is usually less critical than a poor choice of hypotheses, since the inferences become less dependent upon the priors as more data are obtained.

**Sources.** The probabilistic logic shown in Figure 1 and discussed in the surrounding text is due to Aleliunas [2, 3]; actually his axioms are slightly more general than the ones we have listed. Keynes [48], Cox [27], Aczél [1], and Tribus [83] contributed results leading up to this work. A different type of probabilistic logic, in which beliefs are probability ranges rather than point values, was explored by de Finetti [28] and independently by Nilsson [61]. Many techniques for drawing inferences from uncertain information have been devised, for which the book by Pearl [63] is an excellent, recent source.

Bayesian inference is a rich topic with many textbooks and references for both theory and application. As good examples we may cite the statistical texts by Berger [16] and Box and Tiao [22], and the pattern recognition texts by Duda and Hart [30]

---

<sup>8</sup>An improper prior; see footnote 6 above.

and Young and Calvert [91]. That Bayesian inference is no less than a general learning procedure has been noted [26] but not well documented, particularly in the theoretical literature. The book by Pearl [63] is perhaps the most complete central source to date for Bayesian inference techniques in AI. Applications of Bayesian inference abound; a good source for much of this work is the series of annual proceedings of the Maximum Entropy and Bayesian Inference Conference.

Polemics about the comparative merits of Bayesian learning *vis à vis* other methods are a constant source of entertainment, and no one entertains with more insight than Jaynes [40]. Closely related to Bayesian inference are the maximum-entropy method (due to Jaynes) and minimum-cross-entropy method (due to I. J. Good). This relationship is treated formally in [79]. Vapnik [88] discusses Bayesian and other methods for pattern recognition and proves the optimality result cited above.

[41] and [42] were breakthrough papers in the theory of choosing priors. Berger and Berry [17] argue cogently that classical statistical inference (of the Fisher-Neyman-Pearson school) is no less subjective than Bayesian statistics; they suggest that the Bayesian approach of including all subjective information explicitly in the form of priors is preferable to embedding it in the experimental procedure, where it is harder to identify.

## 4 Learnability

Although Bayesian learning is a powerful method for making inferences from sample data, little is known about how computationally difficult such an inference can be. In the past few years a series of formal learning models, often called learnability theory, has been used to study questions such as these:

- How complex is the learning problem in a particular domain? Particularly, what can be learned in (say) polynomial time or logarithmic space?
- If we change to a different representation, does the learning problem become quantifiably easier?
- How can a learning algorithm be designed with provable performance guarantees?

**PAC-Learnability.** The *PAC*-learning<sup>9</sup> model differs from those we have considered until now by explicitly quantifying the running time of the learning algorithm and the accuracy of its result. We continue within the framework of concept learning, although the theory can be applied to other types of learning problems as well. The teacher in the *PAC*-learnability model selects points from the universe  $U$  independently and at random, with probabilities determined by some fixed probability distribution  $P$ ; the teacher then labels each one positive (if in the target concept) or negative (if not). The learner does not know what  $P$  is, and can make no assumptions about the distribution of the training data; hence the results are distribution independent.

---

<sup>9</sup>*PAC* is mnemonic for *probably approximately correct*.

Let  $\mathcal{H} = \{H_1, H_2, \dots\}$  be a family of concepts over  $U$ . The teacher selects a concept  $H_*$  from  $\mathcal{H}$  and an arbitrary probability distribution  $P$  over the set  $U$ . The learner asks the teacher for some number  $m$  of examples. In response the teacher chooses  $m$  points from  $U$ , independently and randomly according to the distribution  $P$ . The teacher then indicates for each point  $x$  whether  $x$  belongs to  $H_*$  and presents the set of labeled points to the learner.

The learner's task is to *approximate* the target concept  $H_*$  in finite time, to a specified accuracy. For any concept  $H_i \in \mathcal{H}$ , let  $H_i \Delta H_*$  be the set  $(H_i - H_*) \cup (H_* - H_i)$ , the symmetric difference of the two concepts  $H_i$  and  $H_*$ . The concept  $H_i$  is said to be an  $\epsilon$ -*approximation* of  $H_*$  if the probability  $P(H_i \Delta H_*)$  is at most  $\epsilon$ . Such will be the case if the likelihood is small ( $\leq \epsilon$ ) that another example from the teacher will be a counterexample to  $H_i$ . The learner may request any number  $m$  of examples, but he must output a hypothesis in  $\mathcal{H}$  that  $\epsilon$ -approximates  $H_*$ .

Note that the learner's success in approximating  $H_*$  is being measured by the same probability distribution used by the teacher to select examples. He is not penalized if his result incorrectly classifies points that occur only rarely. Thus *PAC*-learning mirrors the situation where we all have somewhat different versions of a concept (e.g., a "cup"), but agree on everyday instances of the concept (e.g., a coffeemug is a cup, but a tablespoon is not).

There is, in general, no way to guarantee that the learner will always produce an  $\epsilon$ -approximation to the target concept as long as there is any possibility of drawing a wildly unrepresentative sample. The best we can require is that the learner do so on any run of the algorithm *with high probability*: if the algorithm is executed a large number of times, only a small fraction  $\delta$  of them on average fail to output an  $\epsilon$ -approximation.

Summarizing:

- A *PAC*-learning problem consists of four things: a concept family  $\mathcal{H}$  over  $U$ , two parameters  $\epsilon$  and  $\delta$ , each in the range  $(0, 1)$ , and a teacher. The parameter  $\epsilon$  is the required *accuracy* of, and  $\delta$  the *confidence* in, the learner's output.
- The teacher selects a target concept  $H_* \in \mathcal{H}$  and a probability distribution  $P$ . Both of these are hidden from the learner.
- For every problem instance  $(\epsilon, \delta, H_*, P)$ , an algorithm for the *PAC*-learning problem requests a number of classified examples from the teacher, chooses a hypothesis  $H_i \in \mathcal{H}$ , and halts. The number of examples is called the *sample size*.
- The algorithm solves the problem if

$$\text{Prob}[(H_i \Delta H_*) > \epsilon] < \delta.$$

We say that the family  $\mathcal{H}$  of concept representations is (*PAC*-)*learnable*<sup>10</sup> if there exists an algorithm to solve the *PAC*-learning problem.

<sup>10</sup>The appropriateness of the term *learnable* has been criticized, with justification. Along with other technical terms like *information*, it should be treated only as formal terminology.

Consider a simple example. Suppose  $U$  is the set  $\{0, 1\}^n$  of all Boolean  $n$ -tuples, and that we choose to represent concepts as Boolean formulas consisting of a single monomial. For example, the monomial  $x_2(\neg x_3)$ , which we shall write  $x_2x'_3$ , represents the set of all  $n$ -tuples  $(b_1, \dots, b_n)$  with  $b_2 = 1$  and  $b_3 = 0$ . Boolean variables  $x_i$  are commonly used to encode attributes of the target concept (e.g., “flies”, “eats fish”). Suppose that the teacher selects the concept represented by the monomial  $H_* = x_2x'_3$  as the target, and chooses some probability distribution  $P$  over the  $n$ -tuples. Suppose also that the learning algorithm concludes by choosing  $H = x_1x_2$  for its hypothesis. This hypothesis agrees with the target on all points except those of the form  $(1, 1, 1, *, \dots, *)$  or  $(0, 1, 0, *, \dots, *)$ ; (where  $*$  indicates either 0 or 1). This set of points is  $H\Delta H_*$ . If we sum the probabilities of each of these points and the result  $P(H\Delta H_*)$  is at most  $\epsilon$ , then the learner’s answer is an  $\epsilon$ -approximation to the target. Suppose the algorithm requests lots of examples of the target concept, but by a quirk of probabilistic fate receives many copies of the same example  $-(1, 1, 1, \dots, 1)$ . Even though the probability of this happening may be extremely small, unless the probability of the point  $(1, 1, 1, \dots, 1)$  is zero, it is still a possible sampling event. On the basis of this unrepresentative (and uninformative) sample, the learner may output a hypothesis that is not an  $\epsilon$ -approximation to  $x_2x'_3$ . But this is tolerable, provided this sample and others for which the algorithm does not produce an  $\epsilon$ -approximation occur on any individual run of the algorithm with probability less than  $\delta$ .

**Polynomial-time PAC-learning.** We have defined what it means for a domain to be learnable in the PAC framework, but we may also ask what domains are learnable using only “feasible” computational resources, especially time. According to current jargon, “feasible” means “bounded in running time by a polynomial in the parameters of the problem”. Parameters here include the accuracy  $\epsilon$ , the confidence  $\delta$ , and some measure  $n$  of the problem size (such as the number of Boolean variables in the preceding problem). A family  $\mathcal{H}$  of concept representations is *polynomial-time learnable* if it is learnable by an algorithm whose sample size is bounded by a polynomial in  $n$ ,  $1/\epsilon$ , and  $1/\delta$ , and whose running time is bounded by a polynomial in the size of the sample.<sup>11</sup>

To explore these ideas, consider a concept class  $\mathcal{H}$  of cardinality  $N$ . ( $N$  and  $n$  are usually different.) The following simple procedure, which we call the “filtering algorithm”, is the basis for many PAC-learning algorithms. Request  $m$  examples from the teacher (where the value of  $m$  is still to be determined), and output any concept in  $\mathcal{H}$  that is consistent with all  $m$  examples. The sample size  $m$  depends on  $\epsilon$ ,  $\delta$ , and  $N$ . To compute a value for  $m$ , we reason as follows. If a hypothesis  $H$  is not an  $\epsilon$ -approximation of the target, then the probability that a randomly chosen example will be consistent with  $H$  is no more than  $1 - \epsilon$ , and the probability that all  $m$  examples are consistent with  $H$  is at most  $(1 - \epsilon)^m$ . When  $m = \epsilon^{-1} \ln(N/\delta)$ , we have:

$$(1 - \epsilon)^m \leq e^{-\epsilon m}$$

<sup>11</sup>We also assume that the learning algorithms are uniform for  $n$ , even though for some of the results cited this assumption is not necessary.

$$\begin{aligned}
&= e^{-\ln(N/\delta)} \\
&= \delta/N.
\end{aligned}$$

And since there are at most  $N-1$  concepts that are not  $\epsilon$ -approximations, the probability is less than  $\delta$  that any unacceptable hypothesis will survive the test of consistency with all  $m$  examples. Thus

$$m(N, \epsilon, \delta) = \frac{1}{\epsilon} \ln \frac{N}{\delta} \quad (5)$$

examples suffice to achieve *PAC*-learnability. The filtering algorithm is a poly-time *PAC*-learning procedure, *provided* that the teacher returns examples of polynomial length (in bits) and that the task of finding a consistent hypothesis in  $\mathcal{H}$  can be solved in time polynomial in  $m$ . Whether or not these hold depends on the particular domain.

For the family of Boolean monomials used in the previous example, they do. With  $n$  attributes, there are  $3^n$  monomial hypotheses, so it is infeasible to write them all down and scratch out the ones that disagree with some example. However, we can accomplish much the same thing by tracking each individual attribute. Initially we hypothesize the monomial  $x_1 x'_1 \dots x_n x'_n$ , the null concept. In response to a positive example  $+(b_1, \dots, b_n)$ , for each  $i$ , if  $b_i = 1$  then remove the variable  $x'_i$  from the hypothesis (if it has not already been removed); otherwise, if  $b_i = 0$  then remove  $x_i$ . Negative examples will always be consistent with the current hypothesis, and may thus be ignored. After  $m = O[(n/\epsilon) \ln(1/\delta)]$  examples the resulting monomial satisfies the *PAC*-criteria.

But what about more complex domains where these do not hold? Note that the formula (5) is polynomial in  $\log N$ ; thus as long as  $N$ , the number of hypotheses in  $\mathcal{H}$ , is  $O(2^{\text{poly}(n)})$ , the sample size will be feasible. On the other hand, since there are  $2^{2^n}$  Boolean concepts over  $\{0,1\}^n$ , this algorithm cannot be used to learn the family of arbitrary Boolean concepts. One is tempted to conclude that when  $\log N$  is superpolynomial in  $n$ , the family  $\mathcal{H}$  is not polynomial time learnable. But this is not so, since only a subset of  $\mathcal{H}$  may be sufficient to provide an  $\epsilon$ -approximation to any concept in  $\mathcal{H}$ .

Suppose the problem of finding a consistent formula is not feasible. We cannot conclude that the *PAC*-learning problem on  $\mathcal{H}$  is infeasible, because there may be some other algorithm besides the filtering algorithm that solves the problem in polynomial time.

If neither the cardinality of  $\mathcal{H}$  nor the difficulty of finding a consistent hypothesis determines whether a domain is polynomially *PAC*-learnable, we may ask what does. In [20], Blumer, Ehrenfeucht, Haussler, and Warmuth show that it is not the cardinality  $N$ , but the combinatorial property known as the Vapnik-Chervonenkis (VC) dimension of the family  $\mathcal{H}$  that determines whether a polynomial size sample is likely to filter out all unacceptable hypotheses. Unfortunately space does not permit adequate definition or discussion of this quantity, other than to remark that the VC dimension is at most  $\log_2 N$ , but can be much smaller.<sup>12</sup> In the same paper, the authors also show

<sup>12</sup>For the important, but special, case of Boolean concepts families over  $n$  binary attributes, the same authors note that  $\log N$  is bounded above by a polynomial in  $n$  iff the VC dimension is bounded above by a polynomial in  $n$ . This result also occurs in [57].

that something very close to the consistency problem is the problem that, along with the VC dimension, determines whether the domain is polynomially *PAC*-learnable. Polynomial learnability of  $\mathcal{H}$  is equivalent to the requirement that the VC dimension of  $\mathcal{H}$  be bounded by a polynomial in  $n$  and that there exist a randomized poly-time algorithm taking a set of examples as input and producing, with probability at least  $1/2$ , a hypothesis in  $\mathcal{H}$  consistent with the examples.

**Change of representation.** The learnability formalism also helps to quantify the impact of choosing a particular representation for concepts. A concept class is simply a family of subsets of  $U$ , but often there are many languages that can be used to represent the same family. For example, Boolean concepts — subsets of  $\{0,1\}^n$  — can be represented by arbitrary propositional formulas over  $n$  Boolean variables, or by disjunctive-normal-form formulas (DNF), or by conjunctive-normal-form formulas (CNF), etc. Similarly, concepts over binary strings can be represented by formal grammars, automata, and algebraic expressions.

Practitioners have long known that a mere change of representation can turn a difficult learning problem into an easy one and vice versa. The subset of the natural numbers,  $\{1, 11, 1001, 110011, 1010001, \dots\}$  in binary, is much easier to define in ternary:  $\{1, 10, 100, 1000, 10000, \dots\}$ . Consider also the concept class  $\mathcal{H}$  of concepts that can be represented by disjunctive normal form formulas with at most  $k$  terms. For example,  $x_1 x'_3 x_4 \vee x'_2 x_4 x_8$  is a 2-term DNF formula but not a 1-term DNF. With growth measured by the number  $n$  of variables, this family is not *PAC*-learnable for any fixed  $k > 1$  unless — contrary to conjecture — complexity classes NP and R are identical [64]. But by changing the problem to allow the learner to represent the same family of concepts in a different (and more expressive) language, called  $k$ -CNF (CNF formulas with at most  $k$  literals per conjunct), the class  $\mathcal{H}$  becomes *PAC*-learnable. The reason for this turnabout is as follows. Concepts expressed in  $k$ -term DNF are hard to learn because the consistency problem is NP-hard, even though the requisite sample size is feasibly small. By contrast, the consistency problem for  $k$ -CNF requires only polynomial time [87]; moreover, every  $k$ -term DNF formula has an equivalent  $k$ -CNF formula of about the same size. Thus without any large increase in sample size over that needed for  $k$ -term DNF, we can quickly find a consistent  $k$ -CNF formula  $H'$  that  $\epsilon$ -approximates the target concept. Of course since the  $k$ -term DNF family is properly contained in  $k$ -CNF, the algorithm may produce a  $k$ -CNF hypothesis that does not correspond to any  $k$ -term DNF concept. Nevertheless, by this change of representation we satisfy the requirements of the *PAC*-learning problem: to find an  $\epsilon$ -approximation of the target concept in polynomial time.

To summarize: For a given family  $C$  of possible target concepts (subsets of  $U$ ), there are often many different languages (classes of formulas) for representing the concepts in  $C$ . The minimum requirement for such a language  $\mathcal{H}$  is that every concept in  $C$  be represented by some formula in  $\mathcal{H}$ . It is possible that, for a class  $\mathcal{H}$ , the consistency problem is not tractable. In such cases it may help to change to a representation  $\mathcal{H}'$  whose consistency problem is easier to solve. Let  $\mathcal{H}$  and  $\mathcal{H}'$  be representation languages for a family of concepts over the same set  $U$ . We say that  $\mathcal{H}$  is *PAC*-learnable by  $\mathcal{H}'$  if there is an algorithm that solves the *PAC*-learning problem over

$\mathcal{H}$  by choosing hypotheses from  $\mathcal{H}'$ . For every problem instance  $(\epsilon, \delta, H_* \in \mathcal{H}, \mathcal{P})$ , the algorithm must halt after obtaining some classified examples from the teacher and choosing a hypothesis  $H_i \in \mathcal{H}'$ ; and with probability at least  $1 - \delta$ ,  $\mathbf{P}(H_i \Delta H_*) \leq \epsilon$ . When  $\mathcal{H}$  is *PAC*-learnable by  $\mathcal{H}'$ , then  $\mathcal{H}$  is *PAC*-learnable in accordance with our previous definition. One can easily show that if  $\mathcal{H} \subseteq \mathcal{H}'$  and  $\mathcal{H}'$  is polynomial-time *PAC*-learnable, then  $\mathcal{H}$  is polynomial-time *PAC*-learnable by  $\mathcal{H}'$ .

**Learnability of large concept classes.** Learning  $\mathcal{H}$  by a more expressive representation  $\mathcal{H}'$  may not help if  $\mathcal{H}'$  is *too expressive*. Consider concept classes that are regular sets of binary strings  $\{0, 1\}^*$ . For any set of  $m$  examples, we can easily find a consistent hypothesis in the class of all deterministic finite automata (DFAs) by choosing an automaton that accepts precisely those strings occurring as positive examples. Yet rarely does a simple list of the positive examples qualify as learning, and the likelihood that this list  $\epsilon$ -approximates the target DFA is probably rather small. In AI, this observation goes by the name of the *disjunction problem* [14]: representations that are expressive enough to include disjunctions (unions) of singleton concepts are too expressive because their consistency problem has a trivial solution.

In the terminology of learnability theory, domains with a disjunction problem (including DNF formulas and finite automata), are not *PAC*-learnable because the VC dimension increases too rapidly with the size parameter  $n$ . But here we encounter a serious weakness in our definition of *PAC*-learning: some domains that are not *PAC*-learnable according to the definitions and results cited above are, in actuality, quite learnable!

For example, consider again the class of DFAs accepting a subset of the binary strings  $\{0, 1\}^*$ . The teacher picks a DFA of any size and a distribution over  $\{0, 1\}^*$ , and presents a continual stream of classified examples. Can an algorithm *PAC*-learn this DFA? The VC dimension of the family of DFAs is infinite, so according to the main theorem of [20] it is not learnable from any finite sample size. But consider this algorithm:

1. Let  $H_1, H_2, \dots$  be any enumeration of the DFAs. Set  $i := 1$ .
2. Obtain  $m = \epsilon^{-1} \ln(2/\delta)$  examples, and test  $H_i$  for consistency with this sample.
3. If  $H_i$  disagrees with any example, increase  $i$  by 1, replace  $\delta$  by  $\delta/2$ , and go to step 2.
4. Else write down the DFA  $H_i$  and halt.

It is not hard to show that this algorithm produces a *PAC*-approximation to the target. Thus DFAs *are* learnable. What is more, we have used no special properties of DFAs other than their enumerability and the ability to decide whether a DFA accepts an example string. Hence this argument applies equally to any recursively enumerable class with a decidable membership property, regardless of the VC dimension.

So where have we gone wrong? The problem is that our model of *PAC*-learning requires the learner to decide how many examples to obtain *before* testing any hypotheses.<sup>13</sup> This *a priori* sample size is the quantity that is determined by the VC dimension. Without that artificial requirement, some “unlearnable” classes become learnable and even polynomially learnable [53].

To correct this deficiency in our definitions we revise the learnability model as follows. Let  $\mathcal{C}$  be a family of concepts — subsets of the (countable) set  $U$ . Let  $\mathcal{H}$  be a family of representations for concepts in  $\mathcal{C}$  such that every concept in  $\mathcal{C}$  is represented by at least one hypothesis in  $\mathcal{H}$ . To each  $H \in \mathcal{H}$  we assign an integer-valued measure  $s(H)$  of simplicity, which we call *size*. We assume that  $s(H) > 1$  and that  $s(H)$  is easy to compute for all  $H$ . Let  $\mathcal{H}_s$  be the subset of  $\mathcal{H}$  consisting of all concept representations of size  $s$ ; thus  $\mathcal{H} = \bigcup_{s>0} \mathcal{H}_s$ . A concept  $C \in \mathcal{C}$  *belongs to*  $\mathcal{H}_s$  if  $s$  is the minimum size of any of its representations in  $\mathcal{H}$ ; in this case we write  $s(C) = s$ . An instance of a *PAC<sub>s</sub>-learning problem* is a concept  $C_* \in \mathcal{C}$ , parameters  $\epsilon$  and  $\delta$ , and a teacher.

- The teacher chooses an arbitrary probability distribution  $\mathbf{P}$  over  $U$ , and upon request, obtains a sample point  $x$  according to  $\mathbf{P}$ , classifies it as positive or negative according to  $C_*$ , and presents it to the learner.
- A learner takes the two parameters  $\epsilon$  and  $\delta$ , and outputs a hypothesis  $H \in \mathcal{H}$ , after obtaining a number of examples from the teacher. This number may depend on  $\epsilon$ ,  $\delta$ , and the target  $C_*$ .
- We say that  $\mathcal{C}$  is *PAC<sub>s</sub>-learnable by  $\mathcal{H}$*  if there exists a function  $f(s, 1/\epsilon, 1/\delta)$  and a learner such that, for any problem instance, with probability at least  $1 - \delta$ , the learner requests at most  $f(s(C_*), 1/\epsilon, 1/\delta)$  examples and writes down a hypothesis  $H \in \mathcal{H}$  that  $\epsilon$ -approximates  $C_*$ .
- We say that  $\mathcal{C}$  is *polynomially PAC<sub>s</sub>-learnable by  $\mathcal{H}$*  if it is *PAC<sub>s</sub>-learnable* by an algorithm for which the sample-size function  $f(s, 1/\epsilon, 1/\delta)$  is polynomial in all three arguments, and which runs in time bounded by a polynomial in the size of the sample.

Note that the running time of the learner may increase with the size of the target concept. But since the learner does not know the size  $s(C_*)$  in advance, he may need to keep increasing the sample size “on the fly” as he tests larger hypotheses.

Instead of the filtering algorithm, our prototype for designing efficient learning algorithm is as follows. Let  $m(s)$  be a monotone increasing function with the property that  $m(s)$  is an upper bound on the sample size needed to choose an  $\epsilon$ -approximation to any concept of size at most  $s$ , with confidence  $1 - \delta/2^s$ . ( $m(s)$  depends on the VC dimension of the domain.) Then

1. Initialize  $s = 1$ .

---

<sup>13</sup>This assumption in the definition of *PAC*-learnability and its implications went unnoticed by researchers for nearly two years.

2. Obtain enough additional examples so that a total of  $m(s)$  examples are available.
3. If there exists a consistent hypothesis of size  $\leq s$ , write it down and halt.
4. Otherwise increase  $s$  by one and return to step 2.

Note the modified consistency problem in step 3. We can solve this problem if there is a polynomial-time algorithm to choose a hypothesis of minimum size consistent with the sample. And for many domains, the minimum-size consistency problem is polynomially related to the decision problem in step 3. Alas, for a number of interesting domains — including DNF and DFA — this problem is NP-hard. It may suffice, however, to find a consistent hypothesis *polynomially larger* than minimum, and this easier problem can sometimes be solved in polynomial time even when the minimum-size consistency problem cannot.

For example, an *Occam algorithm* is a procedure that finds a consistent hypothesis of size at most  $s^c m^\alpha$  for some constant  $c \geq 1$  and  $\alpha < 1$ , where  $s$  is the size of the minimum consistent hypothesis. Note the factor  $m^\alpha$ : since  $\alpha < 1$  the size of the resulting hypothesis is strictly smaller than the size of the sample (for all sufficiently large target concepts), so that the disjunction problem is eliminated. Upper bounds on the sample size  $m(s)$  for Occam algorithms have been calculated as a function of the VC dimension [20].

Even with  $PAC_\epsilon$ -learnability some concept classes of particular interest remain hard to learn. For example, consider the class of Boolean concepts over  $\{0, 1\}^n$  represented in DNF. A convenient measure for the size  $s$  of a formula is the number of symbols. Boolean concepts are clearly  $PAC_\epsilon$ -learnable by DNF with an exponential sample size, but what about polynomial  $PAC_\epsilon$ -learnability? For any Boolean concept, there is a minimum-size DNF formula to represent it. Finding a minimum-size DNF formula consistent with a set of examples is NP-complete, so a minimum-size filtering algorithm is unlikely to lead to a poly-time algorithm. There may be an Occam or some other learning algorithm that requires polynomial time and a polynomial size sample, but none is known, and many researchers suspect that DNF is not polynomial  $PAC_\epsilon$ -learnable.

Arbitrary Boolean formulas (not just DNF) are a more compact representation than DNF. Hence the learning problem is more difficult, since the minimum-size formula is smaller and the allowable running time correspondingly shorter. Recently Kearns and Valiant [47] showed that learning Boolean formulas is as hard as solving some number-theoretic problems (factoring Blum integers, deciding quadratic residuosity, etc.), all of them problems conjectured to be computationally infeasible.

The class of “regular” concepts (sets of binary strings accepted by DFAs) is of great practical significance. If we measure the size of a DFA by counting states, then each regular concept has a unique smallest representation as a DFA. We have seen that regular concepts are  $PAC_\epsilon$ -learnable by DFAs, but how complex is the learning problem? Finding the smallest DFA consistent with a set of examples is known to be NP-complete. Moreover, Pitt and Warmuth [66] have shown that learning regular concepts — whether by DFAs, NFAs, regular expressions, or regular grammars — is

as hard as learning Boolean formulas, and recently they extended this to show that even finding a consistent DFA polynomially larger than the minimum is hard. An Occam algorithm is, therefore, unlikely to be found. Moreover, the results of [47] imply that the problems of learning acyclic DFAs and of learning a polynomial-size  $\epsilon$ -approximation to a DFA are both as hard as the number-theoretic problems mentioned above. In short, the evidence is compelling that arbitrary Boolean formulas and finite automata are too general to be  $PAC_{\epsilon}$ -learnable.

**Sources.** The spark for the current interest in learnability theory was the pair of papers by Valiant [87, 86]. Statisticians have studied related models [29, 89]; recognition of the relevance of this work and applying it to concept learning was one contribution of the important paper by Blumer *et al.* [19]; the same paper also contains a proof that classes with infinite VC dimensions are not  $PAC$ -learnable. Others pointed out that this non-learnability property depends on the assumption that the sample size is independent of the target [53]. They showed that without this assumption any recursively enumerable hypothesis class with a decidable membership property is learnable.

To date the theory has accumulated more negative (non-learnability) results than positive. Hardness results for the consistency and other problems are given in [11, 12, 64, 46, 66, 65, 47]. Positive (learnability) results are available for  $k$ -CNF [87],  $k$ -decision lists [67], conjunctive and internal-disjunctive formulas [34], functions [58, 57], and others. Occam algorithms are introduced in [21]. When the learner can query actively, learning possibilities change substantially [13]. Even considering the known results for finite automata [5, 12, 69, 68], context-free grammars [73, 10], propositional Horn sentences [11], and problem-space operator heuristics [59], we have only begun to explore this learning problem. The recent volume [35] is a good source for recent research in learnability and other theoretical topics in learning.

An interesting model needs to be robust in the sense that minor variations in its definitions should leave the principal results intact. The  $PAC$ -model has many minor variations and several major ones; these are compared (and shown to be substantially equivalent) in [36]; see also [4]. An important variation, treated there and in [37], is the *prediction* model. The learner must predict membership of the randomly chosen point in the concept before the teacher informs him of the correct answer. Error is measured by the probability of making an incorrect prediction, as a function of the number of examples. Intuitively, a predictor that improves its prediction accuracy after polynomially many examples must be learning some way to approximate the target. Bounds on how well a predictor can do are closely related to bounds on how well a concept learner can do, independent of the choice of representation. This is especially useful in proving lower bounds for hard-to-learn concept classes.

In another important variation, the impact of errors in the teacher's training data has been examined. Errors can be deliberate (malicious errors) or random (noise), or in between. Errors can affect the choice of a point  $x \in U$  or how the teacher classifies it (+ or -). Good bounds on several kinds of errors are known [45, 6, 75, 80, 52]. Closely related is the case where the target concept cannot be represented exactly by any hypothesis in the hypothesis class. Then the *closest approximating concept* becomes the objective of the learning process. Results on this problem are given in

[89], [53], and [4].

Most of the learnability research has been applied to concept learning, but some *PAC*-learning results are starting to appear in other areas too. Rivest and Schapire [69, 74] study the ability of a robot to model its environment using finite automata. Convergence of stochastic models is discussed in [9, 49, 50]. Sutton analyzes a class of incremental prediction models in [81]. Angluin [13] relates the *PAC*-learning model to that used by Shapiro and others, in which the learner must identify the target exactly while receiving from the teacher counterexamples to his hypotheses. Recently she has shown that DFA's and DNF formulas are not learnable in this model either [12, 8].

## 5 Network Models

Recall that AI has long had two competing representational paradigms:

- Explicit symbolic encoding of knowledge structures, coupled to inference algorithms for using that information; and
- A distributed network of rather simple processor nodes, wherein knowledge is an emergent property of the whole network and not necessarily apparent from its microstructure.

In the fifties, and again in the sixties, the latter, *connectionist* view, was aggressively explored, but the symbolic approach eventually assumed the more prominent part in research. The mid-eighties saw renewed interest in connectionist applications, fueled mainly by the (re-)discovery<sup>14</sup> of a learning algorithm known as *error back-propagation* and its application in several impressive experiments.

Actually this was not the first time that connectionist AI had been reinvigorated by the discovery of a learning algorithm: Rosenblatt's perceptron learning algorithm and the Adaline adaptation algorithm of Widrow and Hoff inspired a burst of research in the early sixties. Learning algorithms are critical to the vision of huge intelligent networks constructed of "dumb" elements, since explicitly programming a massive network of heterogeneous processors is clearly impractical.

A "neural" network is usually represented by a directed graph in which the nodes are associated with simple computational units (threshold logic units, finite-state automata, or the like) and the edges carry numerical messages between nodes, modified by fixed weights assigned to each edge. How can we explain the recurring interest in this model?

- The model is potentially massively parallel. In contrast, symbolic algorithms are often hard to adapt to parallel machines because they are conceived as serial procedures.

---

<sup>14</sup>The error back-propagation technique has apparently been found independently by several researchers, including P. Werbos (1974), D. Parker (1982), Y. LeCun (1985), and D. Rumelhart, G. Hinton, and R. Williams (1986).

- The model is distributed. Knowledge is a global property of the network, not concentrated in the high-information content of a few symbols. Consequently network performance may be less sensitive to local hardware failures and more tolerant of noise in the input.
- In some models the processor elements operate asynchronously and use continuous signals. With real-valued outputs and weights, it is possible to describe the network behavior using differential equations rather than combinatorial mathematics (something particularly appealing to scientists with a background in the physical sciences).
- In some models the network is sparsely connected: individual units communicate with only a relatively small subset of the nodes in the network. This suggests that large networks might be configured automatically by a simple learning algorithm that feeds error information back through the net, inducing local changes in connections or weights.
- The model is related to the Hebbian model of the brain. For some people this compatibility with neural models raises the hope that a simple theory might account for experimental observations about perception and cognition.

In both symbolic and connectionist AI research, the excitement over the potential of the ideas, fueled by festive funding levels, has led to a large body of desultory experimentation. At the same time, the progress of rigorous fundamental research based on formal foundations has been modest. As with learnability theory, many (but not all) of the mathematical results are negative ones. These negative results are valuable for guiding research away from less promising directions; unfortunately they are occasionally misinterpreted as discrediting the entire paradigm.

**Perceptrons.** Perceptrons are a class of linear threshold devices. The name derives from their original use in studying the pattern-recognition problems associated with visual perception. Threshold logic is so closely related to Boolean logic that it would be surprising if many of the learnability results didn't have counterparts in threshold logic. Nevertheless the flavor of perceptron results is different from those of the previous sections. One reason is that the predicates of interest tend to be ones with topological characteristics (convexity, connectivity), and those that remain invariant under certain transformation groups. For example, the predicate  $A(x_1, \dots, x_n)$  might be true on any input pattern containing the letter "A", no matter where it occurs within the input field  $x$ ; thus  $A$  is invariant under translations, rotations, etc.

Formally, the perceptron is defined as follows. Let  $R$  be a set of  $n$  binary inputs (a formalized "retina"). A predicate  $\phi$  over  $R$  is a mapping from assignments  $X$  of the  $n$  input values into  $\{0, 1\}$ , where 1 indicates concept membership (true) and 0 non-membership (false). Let  $\Phi$  be a family of such predicates; a predicate  $F$  is said to be *linear with respect to*  $\Phi$  if there exist integers  $\alpha_\phi$  (one for each  $\phi \in \Phi$ ) and  $\theta$  such that

$$\sum_{\phi \in \Phi} \alpha_\phi \phi(X) > \theta$$

iff  $F(X) = 1$ . The complement of such a function is also considered linear in  $\Phi$ .

The family  $L(\Phi)$  of functions linear with respect to  $\Phi$  is easy to realize in hardware, provided  $\Phi$  is not "too large" and the individual predicates  $\phi \in \Phi$  are not "too complex" (terms to be made precise shortly). If the constant predicate  $I(X) = 1$  is among those in  $\Phi$ , then we can always take  $\theta = 0$ .

Let  $\Phi = \{\phi_1, \phi_2\}$ . The predicate  $\phi_1 \vee \phi_2$  is easy to realize by letting  $\alpha_1 = \alpha_2 = 1$  and  $\theta = 0$ . To represent the predicate  $\phi_1 \wedge \phi_2$ , let  $\alpha_1 = \alpha_2 = 1$  and  $\theta = 1$ . For  $\neg\phi_1$ , set  $\alpha_1 = -1$  and  $\theta = -1$ . In this manner one sees that any logical combination of the predicates  $\Phi$  can be obtained by a threshold network of sufficient depth; perceptrons, however, are limited to a depth of one threshold unit.

The *support* of a predicate  $\phi(X)$  is the smallest set of input units in  $R$  upon which  $\phi$  depends. For example, the predicate  $x_1 \vee (x_1 \wedge x_2)$  has support  $\{x_1\}$ . The class known as the set of *linear threshold functions* is  $L(\Phi)$ , where  $\Phi$  is the set of predicates  $\{x \mid x \in R\}$ . A predicate of the form  $x_{i_1} \wedge \dots \wedge x_{i_k}$  ( $k \leq n$ ) is called a *mask* of order  $k$ . Since any predicate  $F$  can be written in disjunctive normal form, and  $\neg x_i$  can be realized by  $1 - x_i$ , every predicate  $F(X)$  over  $R$  is in  $L(M)$ , where  $M$  is the set of masks. This so-called *positive normal form* for  $F$  is unique. The *order of a predicate*  $F$  is the maximum order of any mask in its positive normal form.

There are  $2^n$  possible masks. Any function whose positive normal form requires a substantial proportion of them surely cannot be considered realizable by perceptrons except for very small  $n$ . For this reason, functions of bounded order are of primary interest. Feasibility also requires that the coefficients  $\alpha$  and  $\theta$  be expressible with a reasonable number of bits. Together, these conditions impose limits on what can be feasibly represented with perceptrons. A well-known result of Minsky and Papert [56] states that the parity ("an odd-number of bits in  $X$  are 1") and connectedness predicates are not computable by finite-order perceptrons. Less familiar, but equally interesting, perceptron results are:

- The parity predicate requires  $\Omega(2^n)$  bits to represent the coefficients  $\alpha$ .
- The counting predicates  $F_m(X) =$  "exactly  $m$  bits of  $R$  are one" are predicates of order 2.
- The "convex-figure" and "rectangular" predicates have order 3.
- Many low-order predicates cease to have bounded order when generalized to detect the property for some connected component of  $X$ . For example, " $R$  consists of a hollow square" is of finite order, but " $R$  contains a hollow square" is not.

Such results, and more significantly, the techniques developed to obtain such results, help to understand the types of pattern concepts that can be represented by perceptrons.

Given that a predicate is in  $L(\Phi)$ , how do we find a set of coefficients  $\alpha_\phi$  for it? (We assume henceforth that  $\theta = 0$ .) The remarkable *perceptron convergence theorem* of Rosenblatt states that a simple, intuitive, linear-feedback algorithm will

eventually converge to a correct set of coefficients. This theorem is noteworthy, not because it is a learning algorithm — after all, direct enumeration of the coefficients will eventually converge, too — but because it is so simple, and because its running time is approximately proportional to the sum of the values of the coefficients it finds. In practice this is much faster on average than identification by enumeration.

A sketch of the algorithm is as follows. Examples are, again, points  $\mathbf{x} = (x_1, \dots, x_n)$  (with  $x_i \in \{0, 1\}$ ), flagged + or - according to whether the target predicate is 1 or 0 on that point. The hypothesis maintained by the algorithm is represented by the set of coefficients  $\alpha_i$ , for all  $1 \leq i \leq n$ . (Recall that  $\theta = 0$ .) The value predicted by the algorithm for the point  $\mathbf{x}$  is the truth value (1 or 0) of the predicate:  $\sum_i \alpha_i \phi_i(\mathbf{x}) > 0$ . The teacher provides a counterexample to the current hypothesis as long as this remains possible. The learning procedure is as follows:

1. Initialize  $\alpha_i = 0$  for  $1 \leq i \leq n$ . (Actually the initial values can be arbitrary.)
2. For each counterexample:
  - 2.1 If the example is positive  $+\mathbf{x}$ , then for each  $i$  such that  $\phi_i(\mathbf{x}) = 1$ , increase  $\alpha_i$  by 1. (Promotion step.)
  - 2.2 If the example is negative  $-\mathbf{x}$ , then for each  $i$  such that  $\phi_i(\mathbf{x}) = 1$ , decrease  $\alpha_i$  by 1. (Demotion step.)

The intuition behind the algorithm is immediate: for each counterexample, those predicates  $\phi_i$  contributing to the error have their coefficient increased (if the false value is 0) or decreased (if 1). The convergence theorem says that after a finite number of counterexamples, the hypothesis will classify all points in  $R$  correctly, assuming that the target hypothesis is in  $L(\Phi)$ . Note that exponentially many counterexamples may be required — e.g., if every perceptron representing the target has an exponentially large coefficient (as does the parity predicate).

As discussed in the section on learnability, contemporary models of pattern discrimination treat predicates over variables  $x$  that represent binary-coded attributes (“is red”, “breathes fire”) rather than pixel activation as in a retina. In such problems, the geometry of the attributes is not a concern, and the predicates of interest are not expected to be invariant under group action. The set  $\Phi$  of basic predicates is viewed as a set of abstract attributes, or *features*, rather than masks. Once the set of features  $\Phi$  has been chosen, the problem of learning  $L(\Phi)$  is formally identical to the perceptron learning problem, and Rosenblatt’s algorithm can be used.

Recently Littlestone [55] has shown how the above learning algorithm can be improved. Whereas the coefficients  $\alpha_i$  in Rosenblatt’s algorithm are incremented by a constant amount (one) for each counterexample, in his algorithm they are *multiplied* by a constant when too small and divided by the constant when too large. As a result the convergence is potentially much faster. The price to be paid is that the program must be given some information about the subclass of linearly separable functions from which the target has been chosen. Call the function  $F \in L(\Phi)$   $\Delta$ -separable if there exist coefficients  $\alpha_i \geq 0$  such that

$$\sum_i \alpha_i \phi_i(\mathbf{x}) > 1 \text{ if } F(\mathbf{x}) = 1,$$

$$\leq 1 - \Delta \text{ if } F(\mathbf{x}) = 0.$$

Then his algorithm converges to a solution whenever the target is  $\Delta$ -separable, after  $O((\log n/\Delta^2) \sum_i \alpha_i)$  counterexamples, where  $n = |\Phi|$ . Note that the resulting coefficients are all positive. Other classes of functions — including those for which negative coefficients may be necessary, and even functions that are not in  $L(\Phi)$  — can be learned by first carrying out a change of representation  $T : \Phi \rightarrow \Phi'$  to a new family of attributes defined in terms of the old, and running the learning algorithm for the new family. The resulting network is a perceptron (over  $\Phi'$ ) preceded by a circuit for carrying out the transformation  $T$ .

Another intriguing property of the Littlestone algorithm is this: when the cardinality of the support (the minimum number of relevant attributes) of the target predicate  $F$  is small compared to  $n$ , the number of counterexamples required before the algorithm converges may also be quite small ( $O(\log n)$ ). Many learning algorithms run in time proportional to the total number of attributes ( $n$ ), even when only a few of those attributes are needed to define the concept. But the number of counterexamples required by Littlestone's algorithm is  $O(\log n \sum_i \alpha_i)$ . When most of the coefficients  $\alpha_i$  are zero and the others are bounded,  $O(\log n)$  passes will be required. Thus for an important class of concepts that depend on only a small subset of a much larger collection of observed features, Littlestone's algorithm finds the target concept quickly by identifying those few relevant features and suppressing the many irrelevant ones.

Linear programming techniques can also be used to solve perceptron learning problems. We can store all the examples the teacher has shown us and treat each example as a constraint:  $\sum_i \alpha_i \phi_i(\mathbf{x}) > 0$  for a positive example,  $\leq 0$  for a negative one. We then minimize  $\sum_i \alpha_i$  using our favorite algorithm (e.g., simplex). But this algorithm stores all examples and constructs each new hypothesis from scratch rather than from the existing one. Thus we are unlikely to regard this "ballistic" algorithm as a learning algorithm, in the sense discussed in the introduction.

**Multi-layer networks.** By now perceptrons are fairly well understood, but the class of concepts that can be represented efficiently with a perceptron is limited. When we generalize in the natural way — by adding one or two additional layers of units between the inputs  $R$  and the output — we find that much less is known about the resulting networks. Comparison is complicated by the fact that most of the problems studied on these networks are *training* problems rather than concept-learning problems.

First some terminology: An acyclic threshold-logic network has *depth*  $k$  when the longest path from an input signal to the output passes through at most  $k$  threshold units (we also say that the circuit has  $k$  *layers*). Thus a perceptron has unit depth. When  $\Phi = \{x_i \in R\}$ , a depth of three suffices to realize every concept on  $R$ ; we shall assume henceforth that  $\Phi$  is this basic set. We also assume that there is a single output unit. Threshold nodes other than the output unit(s) are often called *hidden units*. Both the inputs  $x_i$  and outputs from hidden units are connected to threshold nodes via weighted edges, just as for perceptrons. Weights and thresholds can be arbitrary real numbers (although integers generally suffice). A threshold unit with inputs  $s_i$ , weights  $\alpha_i$ , and a threshold of  $\theta$  outputs 1 if  $\sum_i \alpha_i s_i > \theta$  and 0 otherwise.

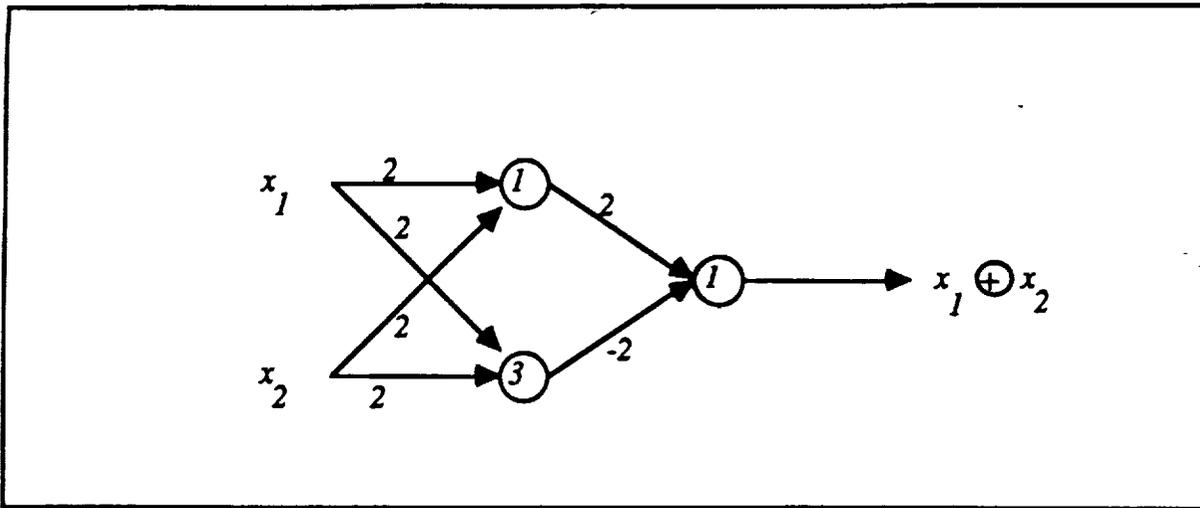


Figure 2: Two-input parity network. Thresholds are written in the nodes, weights beside the edges.

To illustrate, in Figure 5 we show a network with depth 2 that computes the parity function for  $n = 2$ :  $F(x_1, x_2) = 1$  iff  $x_1 \neq x_2$ .

A *training problem* consists of a set  $S$  of examples of some function  $F(x_1, \dots, x_n)$ , with the examples labeled as + or - in the usual way. In general the set  $S$  is a proper subset of all  $2^n$  possible examples of the target  $F$ . The task is to construct a network that agrees with the examples by emitting 1 for each positive example and 0 for each negative example; for inputs not in  $S$  the output may be arbitrary. From the preceding section on learnability theory, we recognize this as a consistency problem, and know that if the examples have been selected randomly, and enough of them are provided, then we have a *PAC-learning* problem. But here we are required only to learn the set  $S$ .

The training problem has been shown to be NP-complete when the network configuration is given and the task is to choose the weights and thresholds. Thus, unlike the simple perceptron training problem (which is essentially a linear programming problem), the multi-layer training problem is unlikely to have any feasible algorithm — incremental or ballistic.<sup>15</sup>

In pursuit of an efficient algorithm we can try relaxing some of the requirements of the training problem. One is to require that it work for only some fraction of the examples; but if that fraction is more than  $2/3$ , the problem remains NP-complete [44]. Another is to replace the linear-threshold function computed at each node by some other function. But if this is still a Boolean-valued function, the problem remains intractable [43].

Suppose the nodes are allowed to compute a non-Boolean valued function. An

<sup>15</sup>D. Haussler (personal communication) points out that, from a *PAC-learning* perspective, the hardness results suggest that multi-layer threshold networks probably cannot learn (feasibly) the entire class of functions that they can represent when examples come from an arbitrary distribution. They may, however, be able to learn a useful subclass, or perhaps the entire class under a restricted set of distributions.

example of such a function frequently used in practice is the so-called *logistic function*

$$f(x_1, \dots, x_n | \alpha_1, \dots, \alpha_n, \theta, \beta) = \frac{1}{1 + \exp[-\beta(\sum \alpha_i x_i - \theta)]}$$

The function  $f$  approaches the standard threshold transfer function as  $\beta \rightarrow \infty$ , but its output is always between zero and one. Complexity results for training with these functions are not available, but there is little reason to suspect that using the logistic transfer function in place of the  $\theta$ -threshold function will reduce the computational complexity of the problem.

Another change that has been widely adopted is to replace the consistency criterion by a *least-mean-square error* (LMSE) procedure. This criterion is attractive in part because it is meaningful for continuous transfer functions, while “consistency” is not. Let  $F(\mathbf{x})$  (where  $\mathbf{x} \equiv (x_1, \dots, x_n)$ ) be the target function and  $H(\mathbf{x})$  the function computed by the current hypothesis machine. The error  $Err$  of  $H$  over the sample  $S$  is given by

$$Err(S) = \sum_{\mathbf{x} \in S} [F(\mathbf{x}) - H(\mathbf{x})]^2.$$

We seek a hypothesis  $H$  for which  $Err(S)$  is minimum.

Apart from any question about the complexity of this optimization problem, we should ask whether this is a good criterion. Of course, without any applications in mind, all such judgments are subjective. But a recent paper [23] provides examples of networks using logistic transfer functions, for which a consistent solution exists, and yet the LMSE solution is inconsistent. Moreover the examples are for networks of unit depth. In response to this and other evidence that the LMSE criterion and associated hill-climbing algorithms may be unsuitable, still more modifications to the models are being studied. But after nearly thirty years of research, an ideal learning algorithm for threshold networks has yet to be found.

**Sources.** Historians will appreciate the view of connectionist AI in the early sixties available from the collection [92]; the paper by Widrow [90] describing the Adaline neural system is of particular interest. Early perceptron work is described in [70] and [60].

The classic work on perceptron theory by Minsky and Papert dates from 1969, but has been reissued with additional commentary [56]. A model of clarity and cogency, this book is as significant today as it was when it first appeared.

Littlestone’s perceptron algorithm is clearly presented in [55] and [54].

A popular source for multi-layer network studies, including the back-propagation algorithm and the Boltzmann machine, is [71]. The intractability of training multi-layer threshold networks was proved by Judd [43, 44]. An extremely simple 2-layer 3-node network that nevertheless is NP-complete to train is described in [18]. See [23] for examples showing that least-mean-squared error algorithms can fail to find solutions when those solutions exist. As I write this, new results are being reported about the complexity of network learning (as distinct from training) in the PAC-learnability model, along with other topics. One can anticipate rapid progress in this field in the coming months and years.

Besides perceptrons, many other network models with learning procedures have been proposed and studied, both experimentally and theoretically. Among these are genetic algorithms [38], "structural" connectionism [32], adaptive resonance [25], and associative search networks [15].

## 6 Afterword

Control theory has produced a rich body of learning research that has not been included in this survey, partly for lack of space and partly because the motivation for this work comes from engineering rather than computer science. Nevertheless this research, often described as "adaptive control systems analysis", probably deserves the attention of learning theorists. Good starting places are the volumes by Tsypkin [84, 85] and the collection [31].

## 7 Acknowledgments

I owe a large debt to the people who took time away from their busy schedules to read drafts of this article, point out errors and offer suggestions: Peter Cheeseman, Silvano Colombano, Peter Dunning, Lol Grant, David Haussler, Nick Littlestone, and John Stutz. Any remaining errors are my responsibility. NASA's Ames Research Center and the AI Research Group under Peter Friedland provided me with the opportunity, incentive, and wherewithal to write this article.

## References

- [1] J. Aczél. *Lectures on Functional Equations and their Applications*. Academic Press, New York, 1966.
- [2] Romas Aleliunas. A new normative theory of probabilistic logic. In *Proceedings, Canadian Society for Computational Studies of Intelligence*, pages 67-74, 1988.
- [3] Romas Aleliunas. A summary of a new normative theory of probabilistic logic. In *Proceedings Uncertainty in AI Workshop*, pages 8-15, 1988.
- [4] J. Amsterdam. Extending the valiant learning model. In *Proceedings, Fifth International Conference on Machine Learning*, pages 381-394, 1988.
- [5] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87-106, 1987.
- [6] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2:343-370, 1987.
- [7] D. Angluin and C. Smith. Inductive inference: theory and methods. *Comput. Surveys*, 15:237-269, 1983.

- [8] Dana Angluin. *Equivalence queries and DNF formulas*. Technical Report YALEU/DCS/RR-659, Yale Univ. Dept. of Comp. Sci., 1988.
- [9] Dana Angluin. *Identifying languages from stochastic examples*. Technical Report YALEU/DCS/RR-614, Yale University Dept. of Computer Science, 1988.
- [10] Dana Angluin. *Learning  $k$ -bounded context-free grammars*. Technical Report YALEU/DCS/RR-557, Yale Univ. Dept. of Comp. Sci., 1987.
- [11] Dana Angluin. *Learning propositional Horn sentences with hints*. Technical Report YALEU/DCS/RR-590, Yale Univ. Dept. of Comp. Sci., 1987.
- [12] Dana Angluin. *Negative results for equivalence queries*. Technical Report YALEU/DCS/RR-648, Yale Univ. Dept. of Comp. Sci., 1988.
- [13] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1987.
- [14] R. Banerji. The logic of learning. In *Advances in Computers*, pages 177 – 216, Elsevier, 1985.
- [15] A. Barto, R. Sutton, and P. Brouwer. Associative search networks: a reinforcement learning associative memory. *Biological cybernetics*, 40(2), 1981.
- [16] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York, 1980.
- [17] James O. Berger and Donald A. Berry. Statistical analysis and the illusion of objectivity. *American Scientist*, 76:159–165, 1988.
- [18] A. Blum and R. Rivest. Training a 3-node neural network is NP-Complete. In *Proceedings, First Workshop on Computational Learning Theory*, Kluwer Academic Press, 1988.
- [19] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proc. 18th Symposium on Theory of Computing*, pages 273–282, ACM, 1986.
- [20] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. *Learnability and the Vapnik-Chervonenkis dimension*. Technical Report UCSC-CRL-87-20, University of California, Santa Cruz, 1987. To appear in *J. ACM*.
- [21] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Inf. Proc. Letters*, 24:377–380, 1987.
- [22] G. Box and G. Tiao. *Bayesian Inference in Statistical Analysis*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1973.
- [23] M. Brady, R. Raghavan, and J. Slawny. Gradient descent fails to separate. In *Proc. 2nd Int. Conf. Neural Networks*, pages 649 – 656, 1988.

- [24] W. Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2), 1988.
- [25] G. Carpenter and S. Grossberg. Art 2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26(3), 1987.
- [26] P. C. Cheeseman. In defense of probability. In *Proc. Ninth IJCAI*, pages 1002–1009, 1985.
- [27] R. T. Cox. Probability, frequency, and reasonable expectation. *American Journal of Physics*, 17:1–13, 1946.
- [28] B. de Finetti. *Theory of Probability*. Wiley, New York, 1974.
- [29] L. Devroye and T. J. Wagner. A distribution-free performance bound in error estimation. *IEEE Trans. Info. Theory*, IT-22:586–587, 1976.
- [30] Richard O. Duda and Peter E. Hart. *Pattern Recognition and Scene Analysis*. Wiley-Interscience, 1973.
- [31] K. S. Narendra (ed.). *Adaptive and Learning Systems: Theory and Applications*. Plenum Press, 1986.
- [32] J. Feldman and D. Ballard. Connectionist models and their properties. *Cognitive Science*, 9:205–254, 1982.
- [33] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [34] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–222, 1988.
- [35] D. Haussler and L. Pitt (eds.). *Proceedings, 1st Computational Learning Theory Workshop*. Morgan Kaufmann, 1988.
- [36] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings, 1st Computational Learning Theory Workshop*, 1988.
- [37] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting  $\{0,1\}$ -functions on randomly drawn points (extended abstract). In *Proceedings, 1st Computational Learning Theory Workshop*, 1988.
- [38] J. H. Holland. Escaping brittleness: the possibilities of general-purpose algorithms applied to parallel rule-based systems. In R. S. Michalski et al., editor, *Machine Learning II*, Morgan Kaufmann, 1986.
- [39] H. Ishizaka. Model inference incorporating generalization. In *Proc. Symp. on Software Science and Engineering*, Kyoto, Sept., 1986.
- [40] E. T. Jaynes. *Papers on Probability, Statistics and Statistical Physics*. Volume 158 of *Synthese Library*, D. Reidel, Boston, 1983.

- [41] E. T. Jaynes. Prior probabilities. *IEEE Transactions on Systems and Cybernetics*, SSC-4(3):227–241, September 1968. (Reprinted in [40]).
- [42] E. T. Jaynes. The well-posed problem. *Foundations of Physics*, 3:447–493, 1973. (Reprinted in [40]).
- [43] J. S. Judd. Learning in networks is hard. In *Proceedings, First International Conference on Neural Networks*, I.E.E.E., 1987.
- [44] J. S. Judd. Learning in neural networks (extended abstract). In *Proceedings, First Workshop on Computational Learning Theory*, Kluwer Academic Press, 1988.
- [45] M. Kearns and M. Li. *Learning in the presence of malicious errors*. Technical Report TR-03-87, Harvard University Aiken Computation Lab, 1987.
- [46] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *Proc. 19th ACM STOC*, 1987.
- [47] M. Kearns and L. Valiant. *Learning Boolean Formulae or finite automata is as hard as factoring*. Technical Report TR 14-88, Harvard University, 1988.
- [48] J. M. Keynes. *A Treatise on Probability*. MacMillan, London, 1921.
- [49] P. Laird. Efficient unsupervised learning. In *Proc. 1st Comput. Learning Theory Workshop*, 1988.
- [50] P. Laird. *Learning a probability distribution efficiently and reliably*. Technical Report RIA-88-10-10-0, NASA-Ames Research Center, AI Research Branch, 1988.
- [51] P. Laird. *Learning by Making Models*. Technical Report RIA-88-4-12-0, NASA-Ames Research Center, AI Research Branch, 1988.
- [52] P. Laird. *Learning from Good and Bad Data*. Kluwer Academic, 1988.
- [53] N. Linial, Y. Mansour, and R. Rivest. Results on learnability and the Vapnik-Chervonenkis dimension (extended abstract). In *Proceedings, 1st Computational Learning Theory Workshop*, 1988.
- [54] N. Littlestone. Learning in a layered network with many fixed-function hidden nodes. In *Proc. 1st International Conference on Neural Nets*, 1987.
- [55] N. Littlestone. Learning quickly when irrelevant attributes abound: a new-linear threshold algorithm. *Machine Learning*, 2:285–318, 1987.
- [56] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. M.I.T. Press, 1988. (expanded edition).
- [57] B. Natarajan. *Learning functions from examples*. Technical Report CMU-RI-TR-87-19, Carnegie-Mellon University Robotics Institute, 1987.
- [58] B. Natarajan and P. Tadepalli. On learning boolean functions. In *Proceedings, 19th ACM STOC*, 1987.

- [59] B. Natarajan and P. Tadepalli. Two new frameworks for learning. In *Proceedings, 5th International Machine Learning Conference*, pages 402–415, 1988.
- [60] N. Nilsson. *Learning Machines*. McGraw-Hill, 1965.
- [61] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1), 1986.
- [62] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: an Introduction to Learning Theory for cognitive and computer scientists*. M.I.T. Press, 1986.
- [63] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, 1988.
- [64] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J.ACM*, 35:965–984, 1988.
- [65] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. 1988. (preprint).
- [66] L. Pitt and M. Warmuth. Reductions among prediction problems: on the difficulty of predicting automata. In *Proc., 3rd Structure in Complexity Theory Workshop*, 1988.
- [67] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(4):229–246, 1987.
- [68] R. L. Rivest and R. E. Schapire. Diversity-based inference of finite automata. In *Proc. 28th FOCS*, 1987.
- [69] R. L. Rivest and R. E. Schapire. A new approach to unsupervised learning in deterministic environments. In *Proc. 4th Workshop on Machine Learning*, pages 364–375, 1987.
- [70] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, D.C., 1961.
- [71] D. Rumelhart, J. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the microstructure of cognition*. M.I.T. Press, 1986. (Two Volumes).
- [72] Y. Sakakibara. *Inductive inference of logic programs based on algebraic semantics*. Technical Report 79, International Institute for Advanced Study of Social Information Science, Fujitsu Ltd., 140 Miyamoto, Numazu, Shizuoka 410-03 Japan, 1987.
- [73] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. In *Proc. 1st Workshop on Computational Learning Theory*, 1988.
- [74] R. E. Schapire. *Diversity-based inference of finite automata*. Technical Report MIT/LCS/TR-413, M.I.T. Lab. for Computer Science, 1988.
- [75] G. Shackelford and D. Volper. Learning  $k$ -DNF with noise in the attributes. In *Proceedings, 1st Computational Learning Theory Workshop*, 1988.

- [76] E. Shapiro. *Algorithmic program debugging*. PhD thesis, Yale University Computer Science Dept., 1982. Published by MIT Press, 1983.
- [77] E. Shapiro. A general incremental algorithm that infers theories from facts. In *Seventh IJCAI*, pages 446–451, IJCAI, 1981.
- [78] E. Shapiro. *Inductive inference of theories from facts*. Technical Report, Yale University Computer Science Dept., No. 192, 1981.
- [79] J. E. Shore and R. W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, IT-26:26–37, 1980.
- [80] R. Sloan. Types of noise for concept learning. In *Proceedings, 1st Computational Learning Theory Workshop*, 1988.
- [81] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
- [82] Y. Takada. *A constructive method for grammatical inference of linear languages based on control sets*. Technical Report 78, International Institute for Advanced Study of Social Information Science, Fujitsu Ltd., 140 Miyamoto, Numazu, Shizuoka 410-03 Japan, 1987.
- [83] M. Tribus. *Rational Descriptions, Decisions, and Designs*. Pergamon Press, Oxford, 1969.
- [84] Ya.Z. Tsytkin. *Adaptation and Learning in Control Systems*. Academic Press, 1971.
- [85] Ya.Z. Tsytkin. *Foundations of the Theory of Learning Systems*. Academic Press, 1973.
- [86] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of IJCAI*, pages 560–566, IJCAI, 1985.
- [87] L. G. Valiant. A theory of the learnable. *C. ACM*, 27:1134–1142, 1984.
- [88] V. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, 1982.
- [89] V. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, 1982.
- [90] B. Widrow. Generalization and information storage in networks of adaline neurons. In Yovits et al., editor, *Self-Organizing Systems*, pages 435 – 448, Spartan Books, 1962.
- [91] T. Y. Young and T. W. Calvert. *Classification, Estimation, and Pattern Recognition*. American Elsevier, 1974.

[92] M. C. Yovits, G. Jacobi, and G. Goldstein (eds.). *Self-Organizing Systems*. Spartan Books, Washington, DC, 1962.

**RIA-88-12-05-3**

*Purposive Discovery Of Mathematical Operator Definitions*

MICHAEL SIMS AND JOHN BRESINA

December 1988

In the context of IL, a discovery system for mathematics, we describe our implementation of a general method (called GPP) for the discovery of mathematical operators. This discovery process is driven by the intended purpose of the created operator. The implementation successfully (re)discovered the correct operator definition for multiplication of Conway numbers. The GPP (Generate, Prune and Prove) method is general with respect to the operator's definition language, the specific operators, and the specified purpose of the operator.

---

**RIA-88-12-05-4**

*Constraint Satisfaction With Delayed Evaluation*

MONTE ZWEBEN AND MEGAN ESKEY

December 1988

This paper describes the design and implementation of a constraint satisfaction system which uses delayed evaluation techniques to provide greater representational power and to avoid unnecessary computation. The architecture used is a uniform model of computation, where each constraint contributes its local information to provide a global solution. We demonstrate the utility of the system by formulating a real-world scheduling problem as a constraint-satisfaction problem..

---

**RIA-89-01-01-03**

*A Study of Knowledge-Based Systems for the Space Station*

PETER FRIEDLAND

January 1989

A rapid turnaround study on the potential uses of knowledge-based systems for Space Station Freedom was conducted from October 1987 through January 1988. Participants included both NASA personnel and experienced industrial knowledge engineers. Major results of the study included five recommended systems for the Baseline Configuration of the Space Station, an analysis of sensor hooks and scars, and a proposed plan for evolutionary growth of knowledge-based systems on the Space Station.

---

✓ **RIA-89-01-07-0**

*A Survey of Computational Learning Theory*

PHILIP LAIRD

January 1989

This paper presents an overview of formal learning theory from four viewpoints: logic, Bayesian inference, learnability theory, and neural networks.

---



# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

|   |  |  |                            |
|---|--|--|----------------------------|
| 1. AGENCY USE ONLY (Leave blank)  | 2. REPORT DATE<br>Dates attached         | 3. REPORT TYPE AND DATES COVERED                         |                            |
| 4. TITLE AND SUBTITLE<br><br>Titles/Authors - Attached  |  | 5. FUNDING NUMBERS                                       |                            |
| 6. AUTHOR(S)  |  | 8. PERFORMING ORGANIZATION REPORT NUMBER<br><br>Attached |                            |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br><br>Code FIA - Artificial Intelligence Research Branch<br>Information Sciences Division |  | 10. SPONSORING / MONITORING AGENCY REPORT NUMBER         |                            |
| 9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br><br>Nasa/Ames Research Center<br><br>Moffett Field, CA. 94035-1000               |  | 11. SUPPLEMENTARY NOTES                                  |                            |
| 12a. DISTRIBUTION / AVAILABILITY STATEMENT<br><br>Available for Public Distribution<br><br><i>Pete Fuedel</i> 5/14/92 BRANCH CHIEF            |  | 12b. DISTRIBUTION CODE                                   |                            |
| 13. ABSTRACT (Maximum 200 words)<br><br>Abstracts ATTACHED  |  |  |                            |
| 14. SUBJECT TERMS   |  |  | 15. NUMBER OF PAGES        |
|   |  |  | 16. PRICE CODE             |
| 17. SECURITY CLASSIFICATION OF REPORT   | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT                  | 20. LIMITATION OF ABSTRACT |

