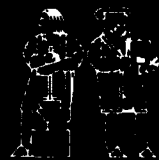LABORATORY FOR
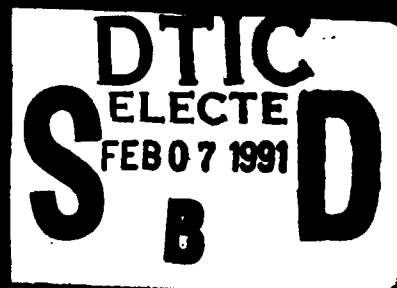COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-493

THE DESIGN AND ANALYSIS OF EFFICIENT LEARNING ALGORITHMS

Robert Elias Schapire

February 1991

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
Unclassified	

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
MIT/LCS/TR 493	

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
MIT Lab for Computer Science		Office of Naval Research/Dept. of Navy

6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code)
545 Technology Square Cambridge, MA 02139	Information Systems Program Arlington, VA 22217

8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS			
1400 Wilson Blvd. Arlington, VA 22217	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

The Design and Analysis of Efficient Learning Algorithms

12. PERSONAL AUTHOR(S)
Robert Elias Schapire

13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) January 1991	15 PAGE COUNT 188

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	machine learning, computational learning theory, concept learning, learning from examples, distribution-free learning probably approximately correct learning, polynomial-time

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Abstract. This thesis explores various theoretical aspects of machine learning with particular emphasis on techniques for designing and analyzing computationally efficient learning algorithms.

Many of the results in this thesis are concerned with a model of concept learning proposed by Valiant.

The thesis begins in Chapter 2 with a proof that any "weak" learning algorithm in this model that performs just slightly better than random guessing can be converted into one whose error can be made arbitrarily small. Several interesting consequences of this result are also described.

Chapter 3 next explores in detail a simple but powerful technique for discovering the structure of an unknown read-once formula from random examples. An especially nice feature of this technique is its powerful resistance to noise.

Chapter 4 considers a realistic extension of the PAC model to concepts that may exhibit uncertain or probabilistic behavior. A range of techniques are explored for designing efficient algorithms for learning such probabilistic concepts.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Carol Nicolora	(617) 253-5894	

18. identification, exact identification, read-once formulas, probabilistic concepts, finite-state automata.

19.   In the last chapter, we present new algorithms for inferring an unknown finite-state automaton from its input-output behavior. This problem is motivated by that faced by a robot in unfamiliar surroundings who must, through experimentation, discover the "structure" of its environment.

     Portions of this thesis are joint work with Sally A. Goldman, Michael J. Kearns and Ronald L. Rivest.

The Design and Analysis of Efficient Learning Algorithms

by

Robert Elias Schapire

S.M., Electrical Engineering and Computer Science
Massachusetts Institute of Technology
(1988)
Sc.B., Mathematics and Computer Science
Brown University
(1986)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1991

© Massachusetts Institute of Technology 1991

Signature of Author_____
Department of Electrical Engineering and Computer Science
January 11, 1991

Certified by_____
Ronald L. Rivest
Professor of Computer Science
Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

The Design and Analysis of Efficient Learning Algorithms

by
Robert Elias Schapire

Submitted to the Department of Electrical Engineering and Computer Science
on January 11, 1991, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis explores various theoretical aspects of machine learning. Particular emphasis is placed on techniques for designing and analyzing computationally efficient learning algorithms.

Many of the results in this thesis are concerned with the so-called *distribution-free* or *probably approximately correct* (PAC) model of learning proposed by Valiant. In this model, the learner tries to identify an unknown concept based on randomly chosen examples of the concept. Examples are chosen according to an unchanging but unknown and arbitrary distribution on the space of instances. The learner's task is to find a hypothesis or prediction rule that, with high probability, correctly classifies all but an arbitrarily small fraction of the instances.

Following a brief introduction, this thesis begins in Chapter 2 with a study of the problem of improving the accuracy of a hypothesis output by a learning algorithm in this model. In particular, it is shown that any "weak" learning algorithm that performs just slightly better than random guessing can be converted into one whose error can be made arbitrarily small. Among the many consequences of this result is a technique for converting any PAC-learning algorithm into one that is highly space efficient.

In Chapter 3, we next explore in detail a simple but seemingly powerful technique for discovering the structure of an unknown read-once formula from random examples. The method is based on sampling of the target formula's statistical behavior under various perturbations of the underlying instance-space distribution. An especially nice feature of this technique is its powerful resistance to noise. One of the highlights of this chapter is the application of this technique to derive the first polynomial-time algorithm for learning read-once Boolean formulas over the usual basis against any product distribution (i.e., any distribution in which the setting of each variable is chosen independently of the settings of the other variables). Algorithms for various other classes of read-once formulas are also presented.

We next consider in Chapter 4 a realistic extension of the PAC model to concepts that may exhibit uncertain or probabilistic behavior. Such *probabilistic concepts* arise naturally in many situations, such as weather prediction, where the measured variables and their accuracy are insufficient to determine the outcome with certainty. While building on the recent results of Haussler on the sample complexity of learning in probabilistic settings, this chapter focuses primarily on the design of efficient algorithms for learning probabilistic concepts. This work also extends many of the results in the standard PAC model to the new probabilistic model.

In the last chapter, we present new algorithms for inferring an unknown finite-state automaton from its input-output behavior. This problem is motivated by the problem faced by a robot in unfamiliar surroundings who must, through experimentation, discover the "structure" of its environment. Some of our algorithms are based on Angluin's algorithm for learning finite-state automata; however, unlike her procedure, our algorithms are effective in the absence of a means of resetting the machine to a start state. We describe provably effective learning algorithms based on both the usual state-based representation, and the *diversity-based* representation in-

troduced by Rivest and Schapire. We also present superior algorithms for the special class of permutation automata.

Portions of this thesis are joint work with Sally A. Goldman, Michael J. Kearns and Ronald L. Rivest.

Thesis Supervisor: Ronald L. Rivest

Title: Professor of Computer Science

Acknowledgments

Publication notes. Much of the research presented in this thesis has appeared elsewhere, in one form or another.

Chapter 2 is a revised version of a paper published with the same title in the journal *Machine Learning* [78]; an extended abstract of this paper also appeared in the proceedings of the *30th Annual Symposium on Foundations of Computer Science*, and an abstract was published in the *Proceedings of the Second Annual Workshop on Computational Learning Theory.*

An extended abstract describing many of the results in Chapter 3 was published with the title "Exact Identification of Circuits Using Fixed Points of Amplification Functions" in the proceedings of the *31st Annual Symposium on Foundations of Computer Science* [30]. This research is joint work with Sally A. Goldman and Michael J. Kearns. An abstract of this research was published in the *Proceedings of the Third Annual Workshop on Computational Learning Theory*, and these results were also included in Sally Goldman's PhD thesis [31].

The research in Chapter 4 is joint work with Michael J. Kearns. An extended abstract describing this research appeared with the same title in the proceedings of the *31st Annual Symposium on Foundations of Computer Science* [53]. An abstract of this research also appeared in the *Proceedings of the Third Annual Workshop on Computational Learning Theory.*

Finally, Chapter 5 is joint work with Ronald L. Rivest and was published in extended abstract form with the same title in the *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing* [74].

Table of Contents

Introduction

The final objective of machine learning research is to build machines that learn from experience. For example, we might like to build an autonomous robot that can adapt to its environment, and can teach itself to walk, navigate, grasp objects, etc. It might also be useful to have a general purpose prediction machine that can study large data bases, recognize patterns, and based on those patterns make predictions about the future. Such a machine might be useful, for instance, as a tool for predicting earthquakes, rendering medical diagnoses, or otherwise aiding in scientific discovery, There is little doubt that machine learning will also play an important role in the development of computers that can speak, read and understand human languages.

The solution of such complicated tasks is clearly beyond the skill of even the best programmer; what is needed is the development of systems that learn, computers that can program themselves. Even if we could directly program a machine to solve such tasks, the result would be a highly inflexible machine capable only of carrying out the specific tasks for which it was pre-programmed. On the other hand, a system incorporating machine learning technology would, in principle, be much more versatile, capable of adapting to changing conditions. Flexibility is vital to several of the problems mentioned above; for instance, a mobile robot should be able to adapt to changing terrain, and a computer that interprets spoken English should be able to adjust to a different speaker.

Besides these practical considerations, the study of machine learning may also lead to a deeper understanding of human intelligence and of the remarkable phenomena of learning, inference, induction and adaptation.

Though the objective of machine learning research seems clear, the way to reach that objective is not so obvious. This tantalizing problem has attracted researchers from a myriad of different fields, including psychology, cognitive science, neural science, linguistics, philosophy,

physics, computer science, control theory and mathematics. The approaches applied to the problem in recent years are also tremendously varied. For instance, there is today a great deal of interest in so-called connectionist learning algorithms [2, 76], some of which are loosely inspired by the anatomy of the brain. Others, such as Holland [45, 46], work on so-called genetic algorithms which learn by mimicking evolutionary processes. Györgi and Tishby [32] have recently used principles of statistical mechanics to understand certain learning algorithms, and Drescher [21] has designed and implemented a learning system based on Piaget's theories of early childhood development. These examples illustrate, but by no means exhaust, the diversity and abundance of approaches applied to the problem of machine learning. For a more comprehensive treatment, see for instance the survey papers of Dietterich [20] and Mitchell et al. [64]; there are also several collections of learning papers available [54, 62, 63, 65, 80].

The results in this thesis belong to an area of machine learning research known as *computational learning theory.* Here, our purpose is to investigate the problem of machine learning in a mathematically-oriented framework. Our goal is to develop a sound, theoretical foundation for studying and understanding machine learning.

Computational learning theory is also characterized by its emphasis on *efficiency.* Thus, we are interested in building machines that not only learn, but that also learn efficiently, consuming limited resources as sparingly as possible. Specifically, we are most often interested in minimizing the time it will take for a machine to learn, the size computer that will be needed (in terms of memory size), and the amount of data that must be collected for learning to take place. Efficiency is an extremely important issue. For the purposes of developing computer programs that run fast, the design of efficient algorithms is arguably of greater importance than the development of faster and more powerful computer processors.

This thesis is about the design of efficient learning algorithms, and the analysis of their performance. The main part of this thesis is organized into four fully self-contained chapters (i.e., no chapter is prerequisite to any other). Each chapter considers a different aspect of machine learning. Here are some of the main contributions:

Chapter 2 describes a general technique for dramatically improving the error rate achieved by a certain kind of concept-learning algorithm. Specifically, it is shown that a "weak" learning algorithm whose error rate is just slightly better than that achieved by random guessing can be converted into one whose error rate is extremely small. A surprising consequence of this result is a technique for dramatically improving the space efficiency of many known learning algorithms.

Chapter 3 explores in detail a simple but powerful statistical method for efficiently inferring the structure of certain kinds of Boolean formulas from random examples of the formula's input-output behavior. A featured application of this method is an algorithm that efficiently

infers a good approximation of any read-once formula (in which each variable occurs at most once) when the random examples are chosen according to a product distribution (in which the setting of each variable is independent of the settings of the other variables).

Chapter 4 extends a standard model of concept learning to accommodate concepts that sometimes exhibit uncertain or probabilistic behavior. This chapter systematically explores a variety of tools and techniques for designing efficient learning algorithms in such a probabilistic setting. For example, we describe an algorithm for learning a probabilistic analog of decision lists.

Finally, Chapter 5 presents a set of efficient algorithms to be used by a robot to infer the "structure" of its environment through experimentation. In particular, we describe algorithms that efficiently infer the structure of any deterministic and finite-state environment by planning and executing experiments, and with the additional aid of a source of "counterexamples" to incorrectly conjectured models of the environment.

A more detailed overview of this thesis follows below.

Concept learning

Many of the results in this thesis deal with one of the most fundamental of learning problems, namely, learning a concept from examples. In recent years, a great deal of research has been devoted to this learning problem, much of it within the framework of one particular learning model introduced by Valiant [83] in 1984. Since it is so important both to the results in this thesis and to current research in computational learning theory, we begin with a brief introduction to the Valiant model.

Informally, a *concept* is a rule that divides the world into positive and negative examples. For instance, the concept of "being red" divides the world into those things that are red and those that are not red. The learning algorithm is presented with examples of the concept, and is told if each is a positive or negative example of the concept. For instance, the learner might be shown several objects and told whether each is red or not.

To determine if the learner has succeeded in "learning" the concept, we give it a test: we present it with one or more unclassified examples, and ask it to predict if each is a positive or negative example of the concept; if the learner "understands" the concept, it should have no trouble passing this test.

The "universe of objects" from which the learner is presented examples is called the *domain* (or sometimes, the *instance space*), and each object in the domain is called an *instance*. For example, in the example above, the domain might consist of all the fruit in the world, in which case all of the examples observed by the learner are pieces of fruit, and the learner's job is to distinguish red fruit from non-red fruit.
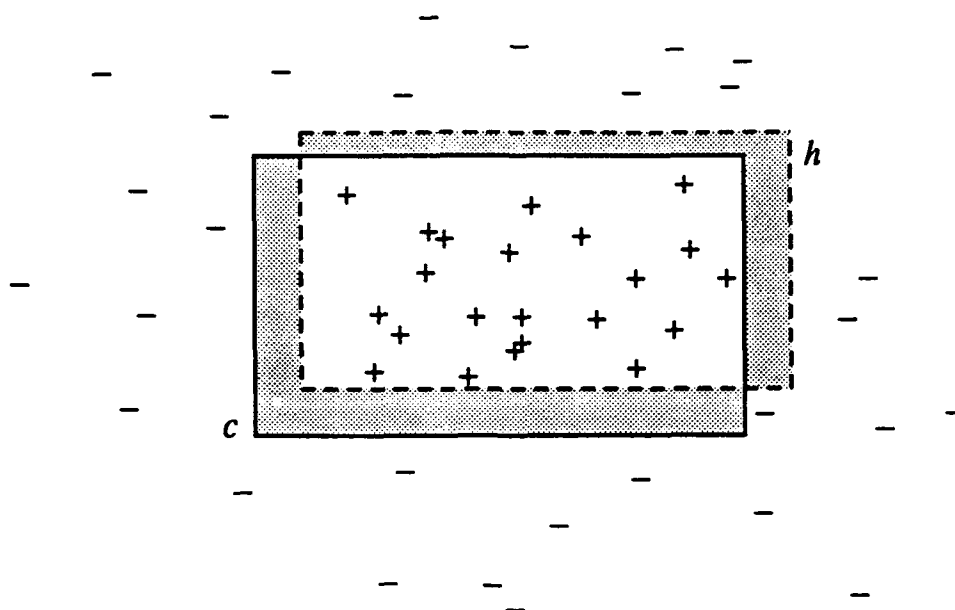
Figure 1: Learning rectangles in the plane.

As another example, the learner might be trying to learn the "concept" of the solid-border rectangle c in Figure 1. In this example, the domain is the set of all points in the real plane. The learner is presented with the sample points labeled $+$ or $-$ as shown in the figure. When tested, the learner may, for example, choose to classify all points inside the dashed-border rectangle h as positive, and those outside as negative. (This would be a reasonable prediction rule since it is *consistent* with the observed sample.) Such a prediction rule is called a *hypothesis*. In this case, the learner will misclassify a test point if and only if it falls in the shaded region, the so-called "symmetric difference" between the two rectangles c and h.

Note that in this example, we assume that the learner knows a priori that the *target concept* c (the one it is trying to learn) is a rectangle. That is, as is typical for such learning problems, the learner knows beforehand that c belongs to some *concept class*, namely, the class of all rectangles.

What we have not yet specified is how the examples presented to the learner for training and for testing should be chosen. Typically, we assume that the examples are chosen at random (although other scenarios are certainly possible). This random selection of examples is meant to model the "random" observations that might be made in the real world. However, the distribution of such observations may be quite arbitrary. We therefore will often ask that our learning algorithm be effective when examples are presented randomly according to *any* distribution on the domain.

Thus, returning to the previous example, we ask that the learner be able to learn the difference between red and non-red fruit, regardless of the "true" distribution of fruit in the world. This may seem unfair since the entirely arbitrary distribution may be such that some kinds of fruit are never observed. For instance, the learner might only be shown green fruit, making it impossible to truly learn the concept of red fruit. However, we only ask that the learner perform well when tested on instances chosen randomly according to the same distribution on which it was trained. Thus, if the learner never saw a red piece of fruit in training, then it is unlikely that it will see one in testing, and so it should be able to pass such a test.

We measure the quality of a learning algorithm by its expected performance on such a test. More precisely, the *error* of a learning algorithm (or rather, of its hypothesis) is the probability that it will misclassify a new instance when, as described above, the new instance is chosen randomly according to the *target distribution*, the distribution on which the learner was trained. Equivalently, the error is the expected fraction of instances that will be misclassified in any test. Similarly, the *accuracy* is the chance that the learner correctly classifies a new instance.

We ask that the learner be able to make its error arbitrarily small. That is, for any arbitrarily small positive number ϵ, the learner should be able to find a hypothesis with error less than ϵ. Also, there is always some small chance that the algorithm receives an unfairly unrepresentative sample that causes it to fail to learn the concept with the desired accuracy. However, we ask that the probability of such a failure be less than δ, where δ, like ϵ, is any arbitrarily small positive number.

Naturally, the smaller the chosen values of ϵ and δ, the larger the sample needed by the learning algorithm, and the longer the computation time needed. However, we ask that the sample size and computation time not grow too quickly as ϵ and δ are made small, nor as other parameters of the learning problem increase. In other words, we ask that the learning algorithm be *efficient*. To make this notion of efficiency more precise, we require that the learning algorithm's running time be bounded by a polynomial in $1/\epsilon$, $1/\delta$ and any other parameters which measure the size of the learning problem. (For example, if learning "hyper-rectangles" in $\mathbb{R}^n$, we might allow the running time to also be polynomial in n.)

As mentioned above, the formal model we have just described was introduced by Valiant [83]. Since the target distribution may be arbitrary, this model is sometimes called the *distribution-free* model. The model is also referred to as the *probably approximately correct (PAC)* model since the learning algorithm's hypotheses should be approximately correct (have low error) with high probability.

Actually, the names "PAC" and "distribution-free" refer to different aspects of the learning model, and it makes sense to talk about a PAC-learning algorithm (one that achieves low error with high probability) that is effective only against certain restricted target distribu-

tions. (However, unless explicitly stated otherwise, PAC learning refers to learning in Valiant's distribution-free model.)

Improving a mediocre learning algorithm

Besides restricting the target distribution, there are many other ways in which the Valiant model might be modified; in fact, most of the results in this thesis deal with variations on the Valiant model. For instance, the Valiant model seems very demanding in its insistence that the learner be able to make the error of its hypotheses arbitrarily small. Why not just require that the learner be 90% correct, or 65% correct? (After all, 65% is a passing grade in most American grade schools.) Note that a learning algorithm that guesses entirely at random on every test point achieves an accuracy rate of 50%. What if we then require the learning algorithm to be only 51% correct? Certainly, we would expect it to be much easier to design such a "weak" learning algorithm that performs just barely better than random guessing than it would be to find one that achieves an accuracy rate of, say, 99.9%.

Chapter 2 of this thesis considers exactly this question. Somewhat surprisingly, it turns out that any such weak learning algorithm can be efficiently converted into one whose error can be made arbitrarily small. This result is relevant to the design of efficient learning algorithms since, in the future, to find an extremely good learning algorithm, it will suffice to first design one that performs only slightly better than random guessing.

The result is relevant to algorithm design for another reason as well: quite unexpectedly, it turns out that the main result of Chapter 2 can be applied to dramatically improve the complexity (i.e., the efficiency) of any PAC-learning algorithm, in several respects. Specifically, we show that *any* PAC-learning algorithm can be converted into one whose time and sample-size requirements come close to the best possible, and, even more surprisingly, whose space (memory) requirements are very modest. For example, the memory size needed by this converted algorithm is much less than would be necessary to store the entire sample (as is done by many previous PAC-learning algorithms).

Learning Boolean formulas from random examples

Thus, dropping the seemingly strong requirement that arbitrarily good error rates be attainable does not change what can or cannot be learned in the Valiant model. However, restricting the target distribution, a variation suggested above, does turn out to significantly affect what can be learned. This is the topic of Chapter 3.

Specifically, Chapter 3 considers the problem of learning read-once Boolean formulas against certain restricted distributions. Informally, a formula is an expression that can be written down in terms of variables and simple functions or operators. A Boolean formula is one in which each

variable is Boolean-valued (i.e., either **true** or **false**), and the formula itself evaluates to a Boolean value.

For example, a car buzzer buzzes if the key is in the ignition and the door is open, or if the motor is on and the seat belt is unfastened. Let K, D, M and S be four Boolean variables which are **true** if and only if, respectively, the key is in the ignition, the door is open, the motor is on, and the seat belt is fastened. Then the buzzer buzzes if and only if the Boolean formula

$$(K \text{ AND } D) \text{ OR } (M \text{ AND NOT}(S))$$

evaluates to **true**. Here, AND, OR and NOT are the standard logical Boolean operators which behave just as would be expected (i.e., x AND y is **true** if and only if both x and y are **true**, and so on). Also, this formula would be said to be *read-once* since it contains at most one occurrence of each variable.

Boolean formulas can be used to compute quite complicated functions, and it is natural to consider their learnability. (Thus, in this learning problem, the domain is the set of all Boolean assignments to the variables, and an instance (an assignment to the variables) is a positive example if and only if this assignment causes the formula to evaluate to **true**.) Unfortunately, Boolean formulas cannot be learned efficiently in the distribution-free learning model (given certain technical assumptions), as was proved by Kearns and Valiant [49, 52]. In fact, their result holds even if a wide range of restrictions are made on the form of the target formula; for instance, their result holds even if the formula is read-once.

Since our goal is to find the most general circumstances in which learning can take place, it makes sense to ask if this intractability result holds even for restricted distributions. For instance, can read-once Boolean formulas be learned efficiently when the target distribution is uniform (assigning equal probability to every point in the domain)? One of the main results of Chapter 3 is an affirmative answer to this question. In fact, we show that such formulas can be PAC learned against any *product distribution* in which the setting of each variable is chosen independently of the settings of the other variables.

This result is based on a simple but powerful technique based on sampling of the formula's statistical behavior under various perturbations of the target distribution. One of the very nice features of this technique is its robustness to noise or randomness that corrupts the learner's observations. The algorithm mentioned above for learning read-once Boolean formulas can handle a great deal of randomness that affects the formula's behavior in a variety of ways.

This technique can also be applied to *exactly identify* certain kinds of read-once formulas against certain fixed distributions; that is, the learning algorithm identifies the exact structure of the target formula, and so obtains a hypothesis with 100% accuracy. Thus, since these same classes of formulas are known to be not even weakly learnable in the distribution-free model,

our results can be interpreted as demonstrating that while there are some distributions which in a computationally bounded setting reveal essentially *no* information about the target formula, there are natural and simple distributions which reveal *all* information.

Dealing with uncertainty

One rather unrealistic aspect of the Valiant model is its assumption of determinism in the classification of instances; that is, we assume that the target concept classifies every instance in the domain as either a positive or a negative example. In the real world, things can be (and usually are) much more uncertain. For example, returning to the problem of learning the concept of "being red," there are many objects in the world which might or might not be called red, depending on who is asked; for instance, this would likely be the case for a crimson Harvard pennant, or a glass of Burgundy. A good learning algorithm should be able to deal with the fact that "red" is a concept with a "fuzzy" boundary, and that some instances in the domain will sometimes be classified "red," and sometimes not.

For another example, consider the problem of learning to predict whether or not it will rain tomorrow based on today's weather conditions. Practically speaking, tomorrow's weather is at least to some degree a probabilistic event, and the best we can usually do is to try to predict the *probability* of rain tomorrow.

Chapter 4 extends the Valiant model to incorporate such *probabilistic concepts* (or *p-concepts*). Specifically, a p-concept c is a real-valued function that assigns to each instance x a probability $c(x)$ of being labeled positively. For instance, the p-concept of "being red" might assign a very high value (near 1) to a strawberry, a very low value (near 0) to a banana, and some value in between to a pomegranate.

This chapter describes efficient algorithms for learning various classes of p-concepts. These include the class of all nondecreasing functions on the real line, and a probabilistic analog of a class of concepts introduced by Rivest [72] called *decision lists*.

In addition to these and other efficient algorithms for learning in the p-concepts model, we study in detail the underlying theory of learning p-concepts. For instance, we give a technique for testing the quality of candidate hypotheses. That is, given two hypotheses, we give a statistical method that can be used to determine which is better. For example, if two meteorologists apply for a job, how can we determine which makes better predictions? If one predicts that it will rain tomorrow with 70% probability, and the other says the chance of rain is 85%, it is not so clear how to determine which prediction is more accurate since tomorrow it will either rain or it won't. (We have no direct access to what the "true" probability of rain is tomorrow.) We describe in this chapter a technique for making such a determination.

We also give a non-trivial lower bound on the number of examples needed to learn in this
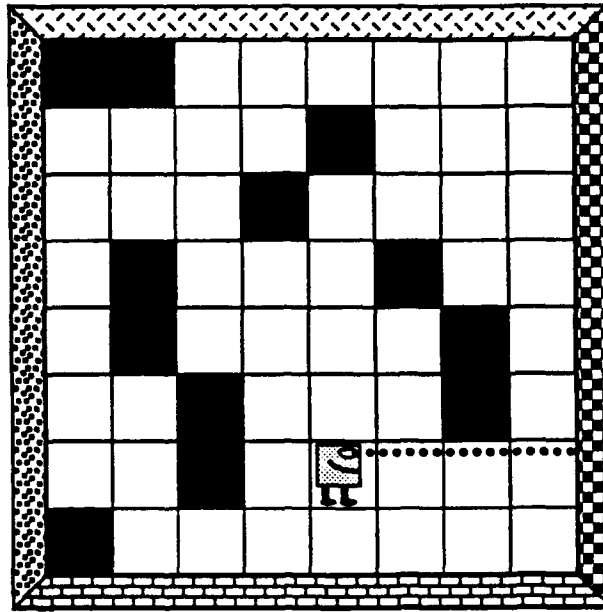
Figure 2: A crossword puzzle environment.

model, and we extend some of the older algorithm-design techniques from the (deterministic) Valiant model to the new p-concept model.

Learning the structure of an unfamiliar environment

Finally, in Chapter 5, we consider a very different learning problem. This chapter, unlike the others, is not concerned with concept learning, nor with learning from random examples. In this chapter, we study the problem of learning about one's environment through experimentation.

Imagine a robot that has been placed in unfamiliar surroundings. For instance, the robot might find itself in the "crossword puzzle" environment of Figure 2. In such an environment, the robot has a limited number of actions, and receives a very limited amount of sensory information about its environment. For instance, in this example environment, the robot can step forward one square, or turn left or right by 90 degrees. If the robot attempts to step forward, but its path is blocked by a wall or one of the black squares, then nothing happens. In this environment, each wall has been painted a different color, and the robot can detect the color of the wall it faces; however, if its view is obstructed by a black square, then it only sees black. This is the only sense data the robot receives about its world.

We assume that the robot knows nothing a priori about its environment, except that the environment is deterministic and finite state. The robot gathers information about its environment by executing actions and observing changes in the sensory data it receives.

We make the realistic assumption that the robot has no "reset" or means of bringing the environment back to some fixed start state. This distinguishes our work from much of the p,evious research on this problem.

The goal of the robot is to discover the "structure" of its environment by planning and executing experiments. In other words, we ask that the robot build a very good model of its environment through experimentation. As was the case for concept learning, we can judge the quality of the robot's model by testing it: given a sequence of actions, the robot should be able to use its model to predict what sensations will be observed when those actions are executed. An action sequence that causes the robot's model to make an incorrect prediction is called a *counterexample*.

It turns out that the robot cannot efficiently build a perfect model of its environment using only experimentation; the reason is that there are some environments with hard-to-reach states that can never be discovered (efficiently) simply through experimentation. We therefore find it necessary to assume that the robot has some source of counterexamples. Thus, the robot runs experiments, builds a model of its environment, and then obtains a counterexample to this conjectured model. The robot repeats this process until it converges to a perfect model. (This source of counterexamples is not as unnatural as it sounds on first blush since, in practice, the robot can often discover counterexamples on its own. For example, a counterexample can often be found by simply taking a "random walk" through the environment.)

In this framework, we describe an efficient algorithm that the robot can use to discover the structure of its environment. This is the first provably fast and effective algorithm for this problem. We also describe an algorithm that solves the same problem, but that is based on a different representation of finite-state environments, called the *diversity-based representation*. Finally, for a special class of environments (called *permutation environments*), we also present efficient algorithms that are effective even in the absence of a source of counterexamples.

Summary

In sum, this thesis extends the current theory of machine learning in several new directions.

Chapter 2 extends our fundamental understanding of the PAC model by demonstrating an important property of this model, namely, that seemingly weak learning algorithms that perform only slightly better than random guessing can be converted into algorithms that perform extremely well. The result also implies interesting and surprising upper bounds on the complexity of learning in the PAC model.

Chapter 3 extends the known techniques for efficiently learning the class of read-once Boolean formulas. The statistical technique that is presented is simple, powerful, and quite robust to noise.

Chapter 4 extends the basic PAC model to incorporate randomness or uncertainty that is almost certain to be found in any real-world application of learning technology. The chapter explores a range of techniques for designing efficient learning algorithms in such a setting.

Finally, Chapter 5 extends our ability to efficiently infer the structure of a deterministic, finite-state environment through experimentation. This chapter includes algorithms for learning in such environments even in the absence of a "reset," using either of two representations for finite-state systems.

The Strength of Weak Learnability

2-1 Introduction

Since Valiant's [83] pioneering paper, interest has flourished in the so-called *distribution-free* or *probably approximately correct (PAC)* model of learning. In this model, the learner tries to identify an unknown concept based on randomly chosen examples of the concept. Examples are chosen according to an unchanging but unknown and arbitrary distribution on the space of instances. The learner's task is to find a hypothesis or prediction rule of its own that correctly classifies new instances as positive or negative examples of the concept. With high probability, the hypothesis must be correct for all but an arbitrarily small fraction of the instances.

Often, the inference task includes a requirement that the output hypothesis be of a specified form. However, in this chapter (and throughout most of this thesis) we will instead be concerned with a representation-independent model of learning in which the learner may output any hypothesis that can be used to classify instances in polynomial time.

A class of concepts is *learnable* (or *strongly learnable*) if there exists a polynomial-time algorithm that achieves low error with high confidence for all concepts in the class. A weaker model of learnability, called *weak learnability*, drops the requirement that the learner be able to achieve arbitrarily high accuracy; a weak learning algorithm need only output a hypothesis that performs slightly better (by an inverse polynomial) than random guessing. The notion of weak learnability was introduced by Kearns and Valiant [52] who left open the question of whether the notions of strong and weak learnability are equivalent. This question was termed the *hypothesis boosting problem* since showing the notions are equivalent requires a method for boosting the low accuracy of a weak learning algorithm's hypotheses.

Kearns [48], considering the hypothesis boosting problem, gives a convincing argument discrediting the natural approach of trying to boost the accuracy of a weak learning algorithm

by simply running the procedure many times and taking the "majority vote" of the output hypotheses. Also, Kearns and Valiant [49, 52] show that, under a uniform distribution on the instance space, monotone Boolean functions are weakly, but not strongly, learnable. This shows that strong and weak learnability are *not* equivalent when certain restrictions are placed on the instance space distribution. Thus, it did not seem implausible that the strong and weak learning models would prove to be inequivalent for unrestricted distributions as well.

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

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

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

Since this result was first published, Freund [26] was able to improve the construction presented in this chapter. His construction yields hypotheses that are simpler in form and smaller in size. Some implementations of his procedure are also more efficient than those described here.

Consequences

An immediate corollary of the main result is the equivalence of strong and *group* learnability. A group-learning algorithm need only output a hypothesis capable of classifying large groups of instances, all of which are either positive or negative. The notion of group learnability was considered by Kearns et al. [51], and was shown to be equivalent to weak learnability by Kearns and Valiant [49, 52]. The result also extends those of Haussler et al. [38] which prove the equivalence of numerous relaxations and variations on the basic PAC-learning model; both weak and group learnability are added to this class of equivalent learning models. The relevance of the main result to a number of other learning models is also considered in this chapter.

An interesting and unexpected consequence of the construction is a proof that any strong learning algorithm outputting hypotheses whose length (and thus whose time to evaluate) depends on the allowed error ϵ can be modified to output hypotheses of length only polynomial in $\log(1/\epsilon)$. Thus, any learning algorithm can be converted into one whose output hypotheses

do not become significantly more complex as the error tolerance is lowered.

Put in other terms, this bound implies that a sequence of labeled examples of a learnable concept can, in a sense, be efficiently "compressed" into a far more compact form—i.e., into a rule or hypothesis consistent with the labels of the examples. In particular, it is shown that a sample of size m can be compressed into a rule of size only poly-logarithmic in m. In fact, in the discrete case, the size of the output hypothesis is entirely independent of m. This provides a partial converse to *Occam's Razor*, a result of Blumer et al. [13] stating that the existence of such a compression algorithm implies the learnability of the concept class. This also complements the results of Board and Pitt [15] who also provide a partial converse to Occam's Razor, but of a somewhat different flavor. Finally, this result yields a strong upper bound on the sample size needed to learn a discrete concept class.

We show that such results also apply to non-discrete domains. In particular, Littlestone and Warmuth [59] describe a notion of data compression in which the output prediction rule is represented by a sequence of examples from the original sample. Such compression schemes are also considered by Floyd [25]. Littlestone and Warmuth show that the existence of an efficient compression scheme of this kind for some concept class implies the learnability of the class. In this chapter, we prove the converse, showing that any learning algorithm can be converted into a compression scheme. Thus, we prove a complete characterization of efficient PAC learnability in terms of data compression.

The bound we prove on the size of the output hypothesis also implies the hardness of learning any concept class not evaluatable by a family of small circuits. For example, this shows that pattern languages—a class of languages considered previously by Angluin [4] and others—are unlearnable assuming only that $NP/poly \neq P/poly$. This is the first representation-independent hardness result not based on cryptographic assumptions. The bound also implies that, for any function not computable by polynomial-size circuits, there exists a distribution on the function's domain over which the function cannot be even roughly approximated by a family of small circuits.

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

larger sample).

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

Translating the bounds described above into the on-line model, it is shown that, for every learnable concept class, there exists an on-line algorithm whose space requirements are quite modest in comparison to the number of examples seen so far. In particular, the space needed on the first m trials is only poly-logarithmic in m. Such space efficient on-line algorithms are of particular interest because they capture the notion of an incremental algorithm forced by its limited memory to explicitly generalize or abstract from the data observed. Also, these results on the space-efficiency of batch and on-line algorithms extend the work of others interested in this problem, including Boucheron and Sallantin [18], Floyd [25], and Haussler [34]. In particular, these results solve an open problem proposed by Haussler, Littlestone and Warmuth [39].

An interesting upper bound is also derived on the expected number of mistakes made on the first m trials. It is shown that. if a concept class is learnable, then there exists an on-line algorithm for the class for which this expectation is bounded by a polynomial in $\log m$. Thus, for large m, we expect an extremely small fraction of the first m predictions to be incorrect. This result answers another open question given by Haussler, Littlestone and Warmuth [39], and significantly improves a similar bound given in their paper (as well as their paper with Kearns [38]) of m^α for some constant $\alpha < 1$.

2-2   Preliminaries

We begin with a description of the distribution-free learning model. A *concept* c is a Boolean function on some domain of *instances*. A *concept class* C is a collection of concepts. Often, C is decomposed into subclasses C_n indexed by a parameter n. That is, $C = \bigcup_{n \geq 1} C_n$, and all the concepts in C_n have a common domain X_n. We assume each instance in X_n has encoded length bounded by a polynomial in n, and we let $X = \bigcup_{n \geq 1} X_n$. Also, we associate with each concept c its *size* s, typically a measure of the length of c's representation under some encoding scheme on the concepts in C.

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

The learner is assumed to have access to a source EX of examples. Each time oracle EX is called, one instance is randomly and independently chosen from X_n according to some fixed

but unknown and arbitrary distribution D. (More formally, D is a probability measure on a σ-algebra of measurable subsets of X_n.) The oracle returns the chosen instance x, along with a label indicating the value $c(x)$ of the instance under the unknown *target concept* $c \in C_n$. Such a labeled instance is called an *example*. We assume EX runs in unit time.

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

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

$$\mathbf{Pr}_{x \in D}[\pi(x)] = \int_{X_n} \mathbf{Pr}[\pi(x)] dD(x).$$

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

We say that a concept class C is *learnable*, or *strongly learnable*, if there exists an algorithm A such that for all $n \geq 1$, for all target concepts $c \in C_n$, for all distributions D on X_n, and for all $0 < \epsilon, \delta \leq 1$, algorithm A, given parameters n, ϵ, δ, the size s of c, and access to oracle EX, runs in time polynomial in n, s, $1/\epsilon$ and $1/\delta$, and outputs a hypothesis h that with probability at least $1 - \delta$ is ϵ-close to c under D. There are many other equivalent notions of learnability, including polynomial predictability [38]. Also, note that other authors have sometimes used the term "learnable" to mean something slightly different.

Kearns and Valiant [52] introduced a weaker form of learnability in which the error ϵ cannot necessarily be made arbitrarily small. A concept class C is *weakly learnable* if there exists a polynomial p and an algorithm A such that for all $n \geq 1$, for all target concepts $c \in C_n$, for all distributions D on X_n, and for all $0 < \delta \leq 1$, algorithm A, given parameters n, δ, the size s of c, and access to oracle EX, runs in time polynomial in n, s and $1/\delta$, and outputs a hypothesis h that with probability at least $1 - \delta$ is $\left(\frac{1}{2} - \frac{1}{p(n,s)}\right)$-close to c under D. In other words, a weak learning algorithm produces a prediction rule that performs just slightly better than random guessing.

2-3   The equivalence of strong and weak learnability

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

Theorem 3.1 *A concept class C is weakly learnable if and only if it is learnable.*

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

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

2-3.1   The hypothesis boosting mechanism

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

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

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

Finally, D_3 is constructed by filtering out from D those instances on which h_1 and h_2 agree. That is, a third oracle EX_3 simulates the choice of an instance according to D_3 by requesting instances from EX until one is found for which $h_1(x) \neq h_2(x)$. (Again, we will later show how to limit the time spent waiting in this loop for a desired instance.) For a third time, algorithm A is simulated with examples drawn this time by EX_3, producing hypothesis h_3.
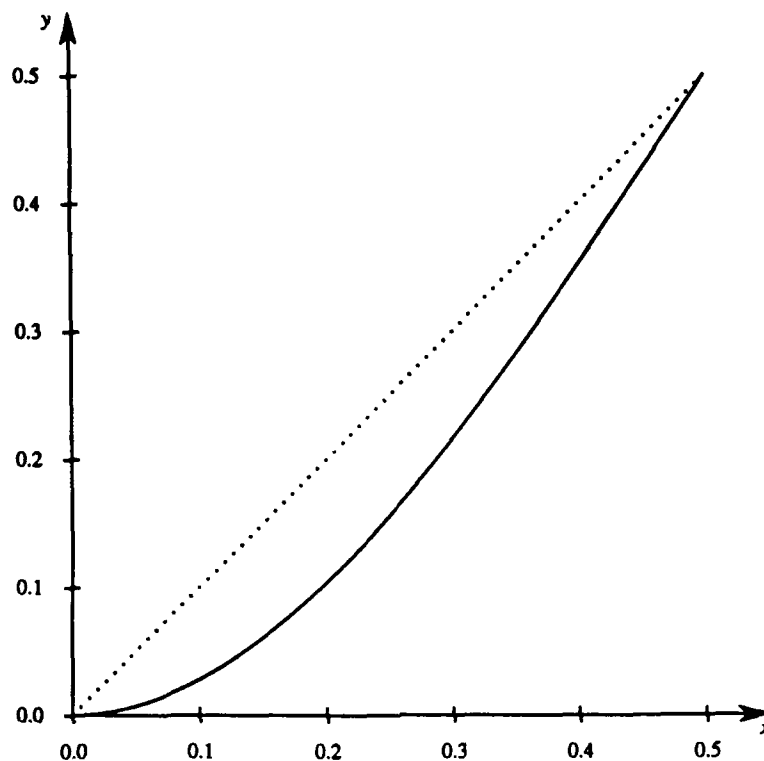
Figure 1: A graph of the function $g(x) = 3x^2 - 2x^3$.

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

2-3.2   A strong learning algorithm

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

Learn(ϵ, δ, EX)

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

Return: hypothesis h that is ϵ-close to the target concept c with probability $\geq 1 - \delta$

Procedure:
     if $\epsilon \geq \frac{1}{2} - \frac{1}{p(n,s)}$ **then return** WeakLearn(δ, EX)
     $\alpha \leftarrow g^{-1}(\epsilon)$

     $EX_1 \leftarrow EX$
     $h_1 \leftarrow$ Learn($\alpha, \delta/5, EX_1$)
     $\tau_1 \leftarrow \epsilon/3$
     let $\hat{a}_1$ be an estimate of $a_1 = \mathbf{Pr}_{x \in D}[h_1(x) \neq c(x)]$:
         choose a sample sufficiently large that $|a_1 - \hat{a}_1| \leq \tau_1$ with probability $\geq 1 - \delta/5$
     if $\hat{a}_1 \leq \epsilon - \tau_1$ **then return** h_1

     defun $EX_2()$
         { flip coin
           if "heads," **return** the first instance x from EX for which $h_1(x) = c(x)$
           else   **return** the first instance x from EX for which $h_1(x) \neq c(x)$ }
     $h_2 \leftarrow$ Learn($\alpha, \delta/5, EX_2$)
     $\tau_2 \leftarrow (1 - 2\alpha)\epsilon/8$
     let $\hat{e}$ be an estimate of $e = \mathbf{Pr}_{x \in D}[h_2(x) \neq c(x)]$:
         choose a sample sufficiently large that $|e - \hat{e}| \leq \tau_2$ with probability $\geq 1 - \delta/5$
     if $\hat{e} \leq \epsilon - \tau_2$ **then return** h_2

     defun $EX_3()$
         { **return** the first instance x from EX for which $h_1(x) \neq h_2(x)$ }
     $h_3 \leftarrow$ Learn($\alpha, \delta/5, EX_3$)

     defun $h(x)$
         { $b_1 \leftarrow h_1(x)$, $b_2 \leftarrow h_2(x)$
           if $b_1 = b_2$ **then return** b_1
           else   **return** $h_3(x)$ }
     return h

Figure 2: A strong learning algorithm **Learn**.

be used to find the desired hypothesis; otherwise, an ϵ-close hypothesis is computed recursively by calling the subroutine with ϵ set to $g^{-1}(\epsilon)$.

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

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

Figure 2 shows a detailed sketch of the resulting strong learning algorithm Learn. The procedure takes an error parameter ϵ and a confidence parameter δ, and is also provided with an examples oracle EX. The procedure is required to return a hypothesis whose error is at most ϵ with probability at least $1 - \delta$. In the figure, p is a polynomial and WeakLearn(δ, EX) is an assumed weak learning procedure that outputs a hypothesis $\left(\frac{1}{2} - \frac{1}{p(n,s)} \right)$-close to the target concept c with probability at least $1 - \delta$. As above, $g(\alpha)$ is the function $3\alpha^2 - 2\alpha^3$, and the variable α is set to the value $g^{-1}(\epsilon)$. Also, the quantities $\hat{a}_1$ and $\hat{e}$ are estimates of the errors of h_1 and h_2 under the given distribution D. These estimates are made with error tolerances τ_1 and τ_2 (defined in the figure), and are computed in the obvious manner based on samples drawn from EX; the required size of these samples can be determined, for instance, using Chernoff bounds. The parameters s and n are assumed to be known globally.

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

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

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

2-3.3   Correctness

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

Theorem 3.2 *For* $0 < \epsilon < 1/2$ *and for* $0 < \delta \leq 1$, *the hypothesis returned by calling* Learn(ϵ, δ, EX) *is* ϵ-*close to the target concept with probability at least* $1 - \delta$.

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

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

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

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

An easy special case is that $\hat{a}_1$ or $\hat{e}$ is found to be smaller than $\epsilon - \tau_1$ or $\epsilon - \tau_2$, respectively. In either case, it follows immediately, due to the accuracy with which a_1 and e are assumed to have been estimated, that the returned hypothesis is ϵ-close to the target concept. (For instance, if $\hat{e} \leq \epsilon - \tau_2$, then $e = \mathbf{Pr}_{x \in D}[h_2(x) \neq c(x)] \leq \epsilon$, and thus the returned hypothesis h_2 is ϵ-close to c.)

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

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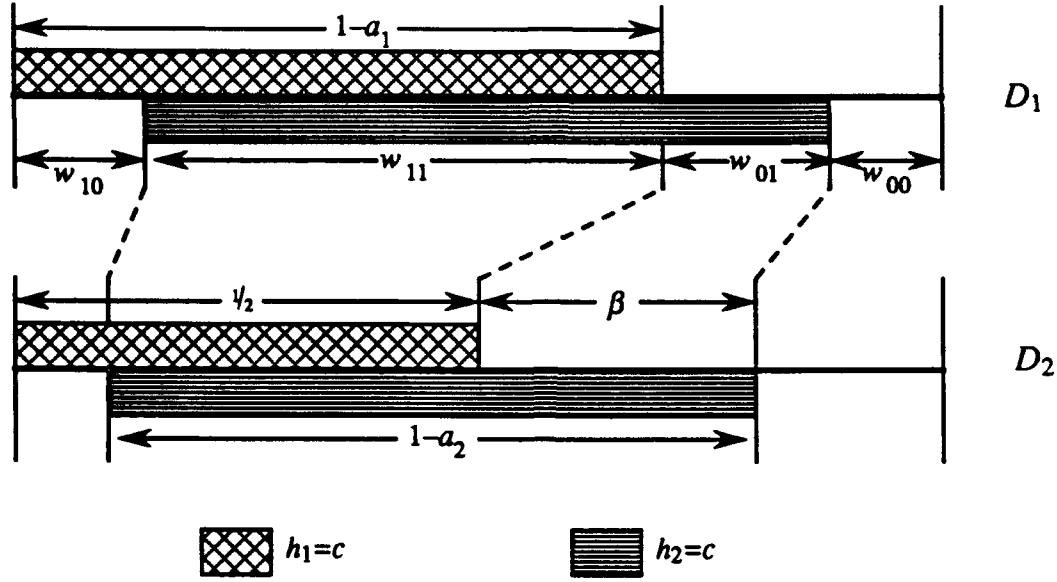
Figure 3: The distributions D_1 and D_2.

Let $p_i(x) = \mathbf{Pr}[h_i(x) \neq c(x)]$ be the chance that some fixed instance x is misclassified by h_i. (Recall that hypotheses may be randomized, and therefore it is necessary to consider the *probability* that a particular fixed instance is misclassified.) Thus, $a_i = \int_{X_n} p_i(x) dD_i(x)$. Similarly, let $q(x) = \mathbf{Pr}[h_1(x) \neq h_2(x)]$ be the chance that x is classified differently by h_1 and h_2. Also define w_{ij} as follows:

$$
\begin{aligned}
w_{00} &= \mathbf{Pr}_{x \in D}[(h_1(x) \neq c(x)) \wedge (h_2(x) \neq c(x))] \\
w_{01} &= \mathbf{Pr}_{x \in D}[(h_1(x) \neq c(x)) \wedge (h_2(x) = c(x))] \\
w_{10} &= \mathbf{Pr}_{x \in D}[(h_1(x) = c(x)) \wedge (h_2(x) \neq c(x))] \\
w_{11} &= \mathbf{Pr}_{x \in D}[(h_1(x) = c(x)) \wedge (h_2(x) = c(x))]
\end{aligned}
$$

Note that the first subscript determines whether h_1 is correct, and the second subscript whether h_2 is correct. Thus, for $i, j \in \{0, 1\}$, we have

$$
w_{ij} = \int_{X_n} |i - p_1(x)| \cdot |j - p_2(x)| dD(x).
$$

Clearly,

$$
w_{00} + w_{01} = \mathbf{Pr}_{x \in D}[h_1(x) \neq c(x)] = a_1, \tag{3.1}
$$

and also

$$w_{00} + w_{01} + w_{10} + w_{11} = 1. \tag{3.2}$$

In terms of these variables, we can express explicitly the chance that EX_i returns an instance from any measurable set $A \subset X_n$:

$$\begin{aligned}
D_1(A) &= D(A) \tag{3.3}\\
D_2(A) &= \tfrac{1}{2}\mathbf{Pr}_{x \in D}[x \in A \mid h_1(x) \neq c(x)] + \tfrac{1}{2}\mathbf{Pr}_{x \in D}[x \in A \mid h_1(x) = c(x)]\\
&= \int_A \left(\frac{p_1(x)}{2a_1} + \frac{1 - p_1(x)}{2(1 - a_1)}\right) dD(x) \tag{3.4}\\
D_3(A) &= \mathbf{Pr}_{x \in D}[x \in A \mid h_1(x) \neq h_2(x)]\\
&= \int_A \frac{q(x)}{w_{01} + w_{10}} dD(x). \tag{3.5}
\end{aligned}$$

In (3.5), we have used the fact that $\mathbf{Pr}_{x \in D}[h_1(x) \neq h_2(x)] = w_{01} + w_{10}$ since c, h_1 and h_2 are Boolean valued.

From equation (3.4), we have that

$$\begin{aligned}
1 - a_2 &= \int_{X_n} (1 - p_2(x)) dD_2(x)\\
&= \int_{X_n} (1 - p_2(x)) \left(\frac{p_1(x)}{2a_1} + \frac{1 - p_1(x)}{2(1 - a_1)}\right) dD(x)\\
&= \frac{1}{2a_1} \int_{X_n} p_1(x)(1 - p_2(x)) dD(x) + \frac{1}{2(1 - a_1)} \int_{X_n} (1 - p_1(x))(1 - p_2(x)) dD(x)\\
&= \frac{w_{01}}{2a_1} + \frac{w_{11}}{2(1 - a_1)}. \tag{3.6}
\end{aligned}$$

To see that the second equality follows from (3.4), see, for instance, Theorem 16.10 of Billingsley [12]. Also, note that (3.6) could have been derived from Figure 3 in the case of deterministic hypotheses: if β is as shown in the figure, then it is not hard to see that $w_{01} = 2a_1\beta$ and $w_{11} = 2(1 - a_1)(1 - a_2 - \beta)$. These equalities imply (3.6).

Combining equations (3.1), (3.2) and (3.6), we see that the values of w_{10} and w_{00} can be solved for and written explicitly in terms of w_{01}, a_1 and a_2:

$$\begin{aligned}
w_{10} &= 1 - a_1 - w_{11}\\
&= 1 - a_1 - 2(1 - a_1)(1 - a_2) + \frac{w_{01}(1 - a_1)}{a_1}\\
&= (2a_2 - 1)(1 - a_1) + \frac{w_{01}(1 - a_1)}{a_1} \tag{3.7}\\
w_{00} &= a_1 - w_{01} \tag{3.8}
\end{aligned}$$

We are finally ready to compute the error of the output hypothesis h. Recall that h is correct if and only if h_1 and h_2 agree in their predictions but are incorrect, or if h_1 and h_2 disagree and h_3 is incorrect. Thus the error of h is

$$
\begin{aligned}
\mathbf{Pr}_{x \in D}[h(x) \neq c(x)] &= \mathbf{Pr}_{x \in D}[[h_1(x) = h_2(x) \neq c(x)] \vee [h_1(x) \neq h_2(x) \wedge h_3(x) \neq c(x)]] \\
&= \mathbf{Pr}_{x \in D}[h_1(x) \neq c(x) \wedge h_2(x) \neq c(x)] \\
&\quad + \mathbf{Pr}_{x \in D}[h_1(x) \neq h_2(x) \wedge h_3(x) \neq c(x)] \\
&= w_{00} + \int_{X_a} q(x)p_3(x)dD(x).
\end{aligned}
$$

By equation (3.5), this is equal to

$$
\begin{aligned}
w_{00} + \int_{X_a}(w_{10} + w_{01})p_3(x)dD_3(x) &= w_{00} + a_3(w_{10} + w_{01}) \\
&\leq w_{00} + \alpha(w_{10} + w_{01}).
\end{aligned}
$$

Applying equations (3.7) and (3.8), this equals

$$
\begin{aligned}
\alpha(2a_2 - 1)(1 - a_1) + a_1 + \frac{w_{01}(\alpha - a_1)}{a_1} &\leq \alpha(2a_2 - 1)(1 - a_1) + \alpha \\
&\leq \alpha(2\alpha - 1)(1 - \alpha) + \alpha = 3\alpha^2 - 2\alpha^3 = g(\alpha) = \epsilon
\end{aligned}
$$

as desired. The inequalities here follow from the facts that each $a_i \leq \alpha < 1/2$, and that, by equation (3.1), $w_{01} \leq a_1$.

This completes the proof.   ∎

A footnote on measurability: For any hypothesis h output by WeakLearn, we have assumed implicitly that the function $f_h(x) = \mathbf{Pr}[h(x) \neq c(x)]$ is measurable. (If this were not the case, then it would hardly make sense to discuss the error of h, since the error is just the expected value of f_h.) By induction, the same can be proved about hypotheses output by Learn: Briefly, each function p_i used in the proof above is D_i-measurable by inductive hypothesis. Thus, by (3.4) and (3.5), p_2 and qp_3 are D-measurable (as follows, for instance, from Billingsley [12] Theorem 16.10). Therefore, the error function $f_h = p_1p_2 + qp_3$ is also D-measurable, where h is the output hypothesis. Note that these facts imply that all the integrals used in the preceding proof are defined.

2-3.4 Analysis

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

of the confidence δ to bound with high probability the actual running time of the algorithm. Thus, we will have shown that the procedure is probably fast and correct, completing the proof of Theorem 3.1. (Although technically we only show that Learn halts probabilistically, using techniques described by Haussler et al. [38], the procedure can easily be converted into a learning algorithm that halts deterministically in polynomial time.)

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

As a technical point, we note that the expectations denoted by T and M are taken only over good runs of Learn. That is, the expectations are computed given the assumption that every subhypothesis and every estimator is successfully computed with the desired accuracy. By Theorem 3.2, Learn will have a good run with probability at least $1 - \delta$.

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

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

Lemma 3.3 *The depth of the recursion induced by calling* Learn(ϵ, δ, EX) *is at most*
$$B(\epsilon, p(n, s)) = O(\log(p(n, s)) + \log\log(1/\epsilon)).$$

Proof: We can say $B(\epsilon, p(n, s)) \leq b + c$ if $g^b \left(\frac{1}{2} - \frac{1}{p(n,s)} \right) \leq 1/4$ and $g^c(1/4) \leq \epsilon$. Clearly, $g(x) \leq 3x^2$ for $x \geq 0$, and so, by an easy induction on i, $g^i(x) \leq (3x)^{2^i}/3$. Thus, $g^c(1/4) \leq \epsilon$ if $c = \left\lceil \lg \log_{4/3}(1/3\epsilon) \right\rceil$.

Similarly, if $1/4 \leq x \leq 1/2$ then $1/2 - g(x) = (1/2 - x)(1 + 2x - 2x^2) \geq (11/8)(1/2 - x)$. This implies (by induction on i) that $1/2 - g^i(x) \geq (11/8)^i (1/2 - x)$, assuming that $x, g(x), \ldots, g^{i-1}(x)$ are all at least $1/4$. Thus, $g^b \left(\frac{1}{2} - \frac{1}{p(n,s)} \right) \leq 1/4$ if $b = \left\lceil \log_{11/8}(p(n, s)/4) \right\rceil$. ∎

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

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

Lemma 3.4 *The time to evaluate a hypothesis returned by* Learn(ϵ, δ, EX) *is*
$$U(\epsilon, \delta) = O(3^B \cdot u(\delta/5^B)).$$

Proof: If $\epsilon \geq \frac{1}{2} - \frac{1}{p}$, then Learn returns a hypothesis computed by WeakLearn. In this case, $U(\epsilon, \delta) = u(\delta)$. Otherwise, the hypothesis returned by Learn involves the computation of at most three subhypotheses. Thus,

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

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

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

■

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

Lemma 3.5 *Let r be the expected number of examples drawn from EX by any oracle EX_i simulated by* Learn *on a good run when asked to provide a single example. Then $r \leq 4/\epsilon$.*

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

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

Finally, when Learn calls itself the third time, we expect the constructed oracle EX_3 to loop $1/(w_{10} + w_{01})$ times before finding a suitable example, since $w_{10} + w_{01}$ is exactly the chance that h_1 and h_2 disagree in their classifications. (Here, the variables w_{ij} are as defined in the proof of Theorem 3.2.) It remains then only to show that $w_{10} + w_{01} \geq \epsilon/4$.

Note that the error e of h_2 on the original distribution D is $w_{10} + w_{00}$. Thus, using this fact and equations (3.1), (3.2) and (3.6), we can solve explicitly for $w_{10} + w_{01}$ in terms of e, a_1 and a_2. Specifically, (3.1) combined with the fact that $e = w_{10} + w_{00}$ gives

$$w_{10} - w_{01} = e - a_1, \tag{3.9}$$

and (3.1), (3.2) and (3.6) together imply that

$$1 - a_2 = \frac{w_{01}}{2a_1} + \frac{1 - a_1 - w_{10}}{2(1 - a_1)}$$

or equivalently,

$$2(1 - a_1)w_{01} - 2a_1 w_{10} = 2a_1(1 - a_1)(1 - 2a_2).$$

Combined with (3.9), this implies

$$(1 - 2a_1)(w_{01} + w_{10}) = e - a_1 + 2a_1(1 - a_1)(1 - 2a_2)$$

and so

$$
\begin{aligned}
w_{01} + w_{10} &= \frac{e - a_1 + 2a_1(1 - a_1)(1 - 2a_2)}{1 - 2a_1} \\
&= a_1 + \frac{e - 4a_1 a_2(1 - a_1)}{1 - 2a_1} \\
&\geq a_1 + \frac{e - 4a_1 \alpha(1 - a_1)}{1 - 2a_1}.
\end{aligned}
\tag{3.10}
$$

Regarding e and $\alpha < 1/2$ as fixed, we refer to this last function on the right hand side of the inequality as $f(a_1)$. To lower bound $w_{10} + w_{01}$, we find the minimum of f on the interval $[0, \alpha]$.

The derivative of f is:

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

The denominator of this derivative is clearly zero only when $a_1 = 1/2$, and the numerator, being a parabola centered about the line $a_1 = 1/2$, has at most one zero less than $1/2$. Thus, the function f has at most one critical point on the interval $(-\infty, 1/2)$. Furthermore, since f tends to $-\infty$ as $a_1 \to -\infty$, a single critical point in this range cannot possibly be minimal. This means that f's minimum on any closed subinterval of $(-\infty, 1/2)$ is achieved at one endpoint of the subinterval. In particular, for the subinterval of interest to us, the function achieves its minimum either when $a_1 = 0$ or when $a_1 = \alpha$. Thus, $w_{10} + w_{01} \geq \min(f(0), f(\alpha))$.

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

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

$$\begin{aligned} &= \frac{\alpha - 6\alpha^2 + 4\alpha^3 + \epsilon}{1 - 2\alpha} \\ &\geq \frac{\alpha - 6\alpha^2 + 4\alpha^3 + (3/4 + \alpha/2)(3\alpha^2 - 2\alpha^3)}{1 - 2\alpha} \\ &= \tfrac{1}{4}\alpha(4 - 7\alpha + 2\alpha^2). \end{aligned}$$

Since $4 - 7\alpha + 2\alpha^2 \geq 1$ for $\alpha \leq 1/2$, $f(\alpha) \geq \alpha/4 \geq \epsilon/4$. We conclude $w_{10} + w_{01} \geq \epsilon/4$, completing the proof. ∎

To bound the number of examples needed to estimate a_1 and e, we make use of the following bounds on the tails of a binomial distribution [10, 44].

Lemma 3.6 (Chernoff Bounds) *Let* $X_1, \ldots, X_m$ *be a sequence of* m *independent Bernoulli trials, each succeeding with probability* p *so that* $\mathbf{E}[X_i] = p$. *Let* $S = X_1 + \cdots + X_m$ *be the random variable describing the total number of successes. Then for* $0 \leq \gamma \leq 1$, *the following hold:*

- *(additive form)* $\mathbf{Pr}[S > (p + \gamma)m] \leq e^{-2m\gamma^2}$, *and* $\mathbf{Pr}[S < (p - \gamma)m] \leq e^{-2m\gamma^2}$;

- *(multiplicative form)* $\mathbf{Pr}[S > (1 + \gamma)pm] \leq e^{-\gamma^2 mp/3}$, *and* $\mathbf{Pr}[S < (1 - \gamma)pm] \leq e^{-\gamma^2 mp/2}$.

The additive form (also known as Hoeffding's inequality) holds also if $X_1, \ldots, X_m$ *are independent identically distributed random variables with range in* $[0, 1]$.

Lemma 3.7 *On a good run, the expected number of examples* $M(\epsilon, \delta)$ *needed by* Learn(ϵ, δ, EX) *is* $O\left(\dfrac{36^B}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right)$.

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

In addition, Learn requires some examples for making its estimates $\hat{a}_1$ and $\hat{e}$. Using the additive form of Lemma 3.6, it follows that a sample of size $O(\log(1/\delta)/\tau_i^2)$ suffices for each estimate, for $i = 1, 2$. Note that $1/p \leq 1/2 - \epsilon = 1/2 - g(\alpha) = (1/2 - \alpha)(1 + 2\alpha - 2\alpha^2) \leq (3/2)(1/2 - \alpha)$. Thus, by our choice of τ_1 and τ_2, both estimates can be made using $cp^2 \log(1/\delta)/\epsilon^2$ examples, for some positive constant c.

We thus arrive at the recurrent inequality:

$$M(\epsilon, \delta) \leq \frac{12}{\epsilon} \cdot M(g^{-1}(\epsilon), \delta/5) + \frac{cp^2 \log(1/\delta)}{\epsilon^2}. \tag{3.11}$$

To complete the proof, we argue inductively that inequality (3.11) implies the bound

$$M(\epsilon, \delta) \leq \frac{36^B \cdot m(\delta/5^B) + c(36^B - 1)p^2 \log(5^B/\delta)}{\epsilon^2}. \tag{3.12}$$

The base case ($B = 0$) clearly satisfies this bound. In the general case, equation (3.11) implies by inductive hypothesis that

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

which clearly implies (3.12). The last inequality here follows from the fact that $\epsilon \leq 3(g^{-1}(\epsilon))^2$ since $g(\alpha) \leq 3\alpha^2$ for $\alpha \geq 0$. ∎

Lemma 3.8 *On a good run, the expected execution time of* Learn(ϵ, δ, EX) *is given by* $T(\epsilon, \delta) =$
$$O\left(3^B \cdot t(\delta/5^B) + \frac{108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B)) \right).$$

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

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

$$T(\epsilon, \delta) \leq 3 \cdot T(g^{-1}(\epsilon), \delta/5) + c \cdot U(g^{-1}(\epsilon), \delta/5) \cdot M(\epsilon, \delta) \tag{3.13}$$

for some positive constant c. Applying Lemmas 3.4 and 3.7, this implies

$$T(\epsilon, \delta) \leq 3 \cdot T(g^{-1}(\epsilon), \delta/5) + \frac{c' \cdot 108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B)) \tag{3.14}$$

for some positive constant c'. An induction argument shows that this implies:

$$T(\epsilon, \delta) \le 3^B \cdot t(\delta/5^B) + \frac{2c' \cdot 108^B \cdot u(\delta/5^B)}{\epsilon^2} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B)) . \qquad (3.15)$$

Clearly, the base case ($B = 0$) satisfies this inequality. In general, equation (3.14) implies using our inductive hypothesis that

$$\begin{aligned}
T(\epsilon, \delta) \le{}& 3^B \cdot t(\delta/5^B) + \\
& \left[\frac{3 \cdot 2c' \cdot 108^{B-1} \cdot u(\delta/5^B)}{g^{-1}(\epsilon)^2} + \frac{c' \cdot 108^B \cdot u(\delta/5^B)}{\epsilon^2} \right] \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B)) .
\end{aligned}$$

Since $g^{-1}(\epsilon) \ge \epsilon$, this clearly implies equation (3.15), completing the induction. ∎

The main result of this section follows immediately:

Theorem 3.9 *Let $0 < \epsilon < 1/2$ and let $0 < \delta \le 1$. With probability at least $1 - \delta$, the execution of Learn($\epsilon, \delta/2, EX$) halts in time polynomial in $1/\epsilon$, $1/\delta$, n and s, and outputs a hypothesis ϵ-close to the target concept.*

Proof: By Theorem 3.2, the chance that Learn($\epsilon, \delta/2, EX$) does not have a good run is at most $\delta/2$. By Markov's inequality, the chance that Learn($\epsilon, \delta/2, EX$) on a good run fails to halt in time $(2/\delta) \cdot T(\epsilon, \delta/2)$ is also at most $\delta/2$. Thus, using Lemma 3.8, the probability that Learn($\epsilon, \delta/2, EX$) has a good run (and so outputs an ϵ-close hypothesis) and halts in polynomial time is at least $1 - \delta$. ∎

2-3.5 Space complexity

Although not of immediate consequence to the proof of Theorem 3.9, it is worth pointing out that Learn's space requirements are relatively modest, as proved in this section.

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

Lemma 3.10 *The space $Q(\epsilon, \delta)$ required to store a hypothesis output by Learn(ϵ, δ, EX) is at most $O(3^B \cdot q(\delta/5^B))$. The space $R(\epsilon, \delta)$ needed to evaluate such a hypothesis is $O(B + r(\delta/5^B))$. Finally, the total space $S(\epsilon, \delta)$ required by Learn is $O(3^B \cdot q(\delta/5^B) + s(\delta/5^B) + B \cdot r(\delta/5^B))$.*

Proof: For $\epsilon \ge \frac{1}{2} - \frac{1}{p}$, the bounds are trivial. To bound Q, note that the hypothesis returned by Learn is a composite of three (or fewer) subhypotheses. Thus,

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

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

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

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

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

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

■

2-4 Improving Learn's time and sample complexity

In this section, we describe a modification to the construction of Section 2-3 that significantly improves Learn's time and sample complexity. In particular, we improve these complexity measures by roughly a factor of $1/\epsilon$, giving bounds that are linear in $1/\epsilon$ (ignoring log factors). These improved bounds will have some interesting consequences, described in later sections.

In the original construction of Learn given in Figure 2, much time and many examples are squandered by the simulated oracles EX_i waiting for a desirable instance to be drawn. Lemma 3.5 showed that the expected time spent waiting is $O(1/\epsilon)$. The modification described below will reduce this to $O(1/\alpha) = O(1/\sqrt{\epsilon})$. (Here, $\alpha = g^{-1}(\epsilon)$ as before.)

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

2-4.1 The modifications

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

$\hat{a}_1 \leq 3\alpha/4$. (For if $\hat{a}_1 > 3\alpha/4$, then $a_1 \geq \alpha/2$ (assuming $\hat{a}_1$ has the desired accuracy); thus, in this case, $|a_1 - 3\alpha/4| \leq \alpha/4$ and $\hat{a}_1$ can be replaced by $3\alpha/4$.)

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

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

If the outcome of this coin flip is "tails," then h_1' evaluates $h_1(x)$ and returns the result. Otherwise, if "heads," h_1' predicts the wrong answer, $\neg c(x)$. Since h_1' will only be used during the training phase, we can assume that the correct classification of x is available, and thus that h_1' can be simulated. Also, note that $0 \leq \beta \leq 1$ since $\hat{a}_1 \leq 3\alpha/4$.

This new hypothesis h_1' is now used in place of h_1 by EX_2 and EX_3. The rest of the procedure **Learn** is unmodified. In particular, the final returned hypothesis h is unchanged—that is, h_1, not h_1', is used by h.

One other modification is made to improve **Learn**'s time and sample complexity: after h_2 is computed, its error e with respect to D is estimated in a slightly different manner. Specifically, we make an estimate $\hat{e}$ with the following properties:

- if $e < \epsilon - 2\tau_2$, then $\hat{e} \leq \epsilon - \tau_2$ with probability at least $1 - \delta/5$; and

- if $e \geq \epsilon - 2\tau_2$, then $\hat{e} \geq (1 - \tau_2/\epsilon)e$ with probability at least $1 - \delta/5$.

We will see that such an estimate has all of the needed properties, but requires a significantly smaller sample.

2-4.2   Correctness

To see that **Learn'** is correct, we will assume as in the proof of Theorem 3.2 that a good run occurs; this will be the case with probability at least $1 - \delta$. If $\hat{e} \leq \epsilon - \tau_2$ so that $h = h_2$ is returned, then either $e < \epsilon - 2\tau_2$, or $\hat{e}$ is such that $e \leq \epsilon \cdot \hat{e}/(\epsilon - \tau_2)$. In either case, the returned hypothesis h_2 clearly has error $e \leq \epsilon$.

Otherwise, note that the error of h_1' is exactly $a_1' = (1 - \beta)a_1 + \beta$ since the chance of error is a_1 on "tails," and is 1 on "heads." By our choice of β, we have that:

$$\begin{aligned} a_1' &\leq (1 - \beta)(\hat{a}_1 + \tfrac{1}{4}\alpha) + \beta \\ &= \hat{a}_1 + \tfrac{1}{4}\alpha + (1 - \hat{a}_1 - \tfrac{1}{4}\alpha)\beta \\ &= \hat{a}_1 + \tfrac{1}{4}\alpha + \tfrac{3}{4}\alpha - \hat{a}_1 = \alpha, \end{aligned}$$

and also

$$
\begin{aligned}
a_1' &\geq (1-\beta)(\hat{a}_1 - \tfrac{1}{4}\alpha) + \beta \\
&= \hat{a}_1 - \tfrac{1}{4}\alpha + (1 - \hat{a}_1 + \tfrac{1}{4}\alpha)\beta \\
&\geq \hat{a}_1 - \tfrac{1}{4}\alpha + (1 - \hat{a}_1 - \tfrac{1}{4}\alpha)\beta \\
&= \hat{a}_1 - \tfrac{1}{4}\alpha + \tfrac{3}{4}\alpha - \hat{a}_1 = \tfrac{1}{2}\alpha.
\end{aligned}
$$

Thus, $\alpha/2 \leq a_1' \leq \alpha$.

Let h' be the same hypothesis as h, except with h_1' used in lieu of h_1. Note that h', h_1', h_2 and h_3 are related to one another in exactly the same way that h, h_1, h_2 and h_3 are related in the original proof of Theorem 3.2. That is, if we imagine that h_1' is returned on the first recursive call of the original procedure Learn, then it is not impossible that h_2 and h_3 would be returned on the second and third recursive calls, in which case h' would be the returned hypothesis. Put another way, the proof that h' has error at most $g(\alpha) = \epsilon$ is an identical copy of the one given in the proof of Theorem 3.2, except that all occurrences of h and h_1 are replaced by h' and h_1'.

Finally, we must show that h's error is at most that of h'. Let $p_1'(x) = \mathbf{Pr}[h_1'(x) \neq c(x)]$, and let $p_i(x)$ be as in Theorem 3.2. Then for $x \in X_n$, we have

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

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

2-4.3   Analysis

Next, we show that Learn' runs faster using fewer examples than Learn. We use essentially the same analysis as in Section 2-3.4. The following three lemmas are modified versions of Lemmas 3.5, 3.7 and 3.8. The proofs of the other lemmas apply immediately to Learn' with little or no modification, and so are omitted.

Lemma 4.1 *Let r be the expected number of examples drawn from EX by any oracle EX_i simulated by* Learn' *on a good run when asked to provide a single example. Then $r \leq 4/\alpha$.*

Proof: As in the original proof, $r = 1$ for EX_1. We expect the second oracle EX_2 to loop at most $1/a_1'$ times on average. Since, as noted above, $\alpha/2 \leq a_1' \leq \alpha$, r is at most $2/\alpha$ in this case.

Finally, to bound the number of iterations of EX_3, we will show that $w_{10} + w_{01} \geq \alpha/4$ using equation (3.10) as in the original proof. To lower bound $w_{10} + w_{01}$, we find the minimum of the last formula f of (3.10) (with a_1 replaced by a_1' of course) on the interval $[\alpha/2, \alpha]$. As noted previously, the function f must achieve its minimum at one endpoint of the interval. We assume as in the original proof that $\hat{e} > \epsilon - \tau_2$ and thus that $e \geq \epsilon - 2\tau_2 = (3/4 + \alpha/2)\epsilon$. It was previously shown that $f(\alpha) \geq \alpha/4$, and, by a similar argument, we can bound $f(\alpha/2)$. Specifically, since $e \geq (3/4 + \alpha/2)(3\alpha^2 - 2\alpha^3)$, we have that

$$
\begin{aligned}
f(\alpha/2) &= \frac{\alpha}{2} + \frac{e - \alpha^2(2 - \alpha)}{1 - \alpha} \\
&\geq \frac{\alpha}{2} + \alpha^3 + \frac{\alpha^2}{4(1 - \alpha)} \geq \frac{\alpha}{2}.
\end{aligned}
$$

This completes the proof.   ∎

Lemma 4.2 *On a good run, the expected number of examples $M(\epsilon, \delta)$ needed by* Learn'(ϵ, δ, EX) *is* $O\left(\dfrac{36^B}{\epsilon} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right)$.

Proof: The proof is nearly the same as for Lemma 3.7. In addition to incorporating the superior bound given by Lemma 4.1 on the number of examples needed by the simulated oracles, we must also consider the number of examples needed to estimate a_1 and e. The first, a_1, can be estimated using a sample of size $O(\log(1/\delta)/\alpha^2) = O(\log(1/\delta)/\epsilon)$; this can be derived using the additive form of Lemma 3.6, and by noting that $\epsilon = g(\alpha) \leq 3\alpha^2$ for $\alpha \geq 0$.

Using the multiplicative form of Chernoff bounds, we argue that e can be estimated with the desired accuracy using a sample of size only $O(p^2 \log(1/\delta)/\epsilon)$. First, if $e < \epsilon - 2\tau_2$, then the chance that $\hat{e}$ exceeds $\epsilon - \tau_2$ when derived from a sample of size m is at most the probability of more than $(\epsilon - \tau_2)m$ successes occurring in a sequence of m Bernoulli trials, each succeeding with probability exactly $\epsilon - 2\tau_2$. Applying the multiplicative form of Lemma 3.6 (with γ set to $\tau_2/(\epsilon - 2\tau_2)$), it follows that this probability is at most $\delta/5$ for m proportionate to $p^2 \log(1/\delta)/\epsilon$. Thus, for such a choice of m, if $e < \epsilon - 2\tau_2$, then $\hat{e} \leq \epsilon - \tau_2$ with probability at least $1 - \delta/5$.

If $e \geq \epsilon - \tau_2$ then, again applying the multiplicative form of Chernoff bounds (with γ set to τ_2/ϵ), we see that $\hat{e} \geq (1 - \tau_2/\epsilon)e$ with probability at least $1 - \delta/5$ for a sample of size proportionate to $p^2 \log(1/\delta)/\epsilon$.

Thus, we arrive at the recurrence

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

for some positive constant c. This implies the stated bound by an argument similar to that given in the proof of Lemma 3.7.   ∎

Lemma 4.3 *On a good run, the expected execution time of* Learn$'(\epsilon, \delta, EX)$ *is given by* $T(\epsilon, \delta) =$
$$O\left(3^B \cdot t(\delta/5^B) + \frac{108^B \cdot u(\delta/5^B)}{\epsilon} \cdot (p^2 \log(5^B/\delta) + m(\delta/5^B))\right).$$

Proof: This bound follows from the recurrence (3.13), using the superior bound on M given by Lemma 4.2. ■

2-5   Variations on the learning model

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

2-5.1   Group learning

An immediate consequence of Theorem 3.1 concerns *group learnability*. In the group learning model, the learner produces a hypothesis that need only correctly classify large groups of instances, all of which are either positive or negative examples. Kearns and Valiant [49, 52] prove the equivalence of group learning and weak learning. Thus, by Theorem 3.1, group learning is also equivalent to strong learning.

2-5.2   Miscellaneous PAC models

Haussler et al. [38] describe numerous variations on the basic PAC model, and show that all of them are equivalent. For instance, they consider randomized versus deterministic algorithms, algorithms for which the size s of the target concept is known or unknown, and so on. It is not hard to see that all of their equivalence proofs apply to weak learning algorithms as well (with one exception described below), and so that any of these weak learning models are equivalent by Theorem 3.1 to the basic PAC-learning model.

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

therefore weak) learnability. Thus, despite the inapplicability of Haussler et al.'s original proof, all four learning models are equivalent.

2-5.3   Fixed hypothesis spaces

Much of the PAC-learning research has been concerned with the form or representation of the hypotheses output by the learning algorithm. Clearly, the construction described in Section 2-3 does not in general preserve the form of the hypotheses used by the weak learning algorithm. It is natural to ask whether there exists any construction preserving this form. That is, if concept class C is weakly learnable by an algorithm using hypotheses from a class $\mathcal{H}$ of representations, does there then exist a strong learning algorithm for C that also only outputs hypotheses from $\mathcal{H}$?

In general, the answer to this question is "no" (modulo some relatively weak complexity assumptions). As a simple example, consider the problem of learning k-term DNF formulas using only hypotheses represented by k-term DNF. (A formula in disjunctive normal form (DNF) is one written as a disjunction of terms, each of which is a conjunction of literals, a literal being either a variable or its complement.) Pitt and Valiant [68] show that this learning problem is infeasible if RP $\neq$ NP for k as small as 2.

Nevertheless, the weak learning problem is solved by the algorithm sketched below. (A similar algorithm is given by Kearns [48].) First, choose a "large" sample. If significantly more than half of the examples in the sample are negative (positive), then output the "always predict negative (positive)" hypothesis, and halt. Otherwise, we can assume that the distribution is roughly evenly split between positive and negative examples. Select and output the disjunction of k or fewer literals that misclassifies none of the positive examples, and the fewest of the negative examples.

We briefly argue that this hypothesis is, with high probability, $(\frac{1}{2} - \Omega(\frac{1}{n^k}))$-close to the target concept. First, note that the target k-term DNF formula is equivalent to some k-CNF formula [68]. (A formula in conjunctive normal form (CNF) is one written as the conjunction of clauses, each clause a disjunction of literals. If each clause consists of only k literals, then the formula is in k-CNF.) Next, we observe that every clause is satisfied by every assignment that satisfies the entire k-CNF formula. Moreover, since the formula has at most $O(n^k)$ clauses, by an averaging argument, there must be one clause not satisfied by $\Omega(1/n^k)$ of the assignments (as weighted by the target distribution) that do not satisfy the entire formula. Thus, there exists some disjunction of k literals that is correct for nearly all of the positive examples and for at least $\Omega(1/n^k)$ of the negative examples. In particular, the output hypothesis has this property. Since the distribution is roughly evenly divided between positive and negative examples, this implies that the output hypothesis is roughly $(\frac{1}{2} - \Omega(\frac{1}{n^k}))$-close to the target formula.

2-5.4   Queries

A number of researchers have considered learning scenarios in which the learner is not only able to passively observe randomly selected examples, but is also able to ask a "teacher" various sorts of questions or *queries* about the target concept. For instance, the learner might be allowed to ask if some particular instance is a positive or negative example. Angluin [7] describes several kinds of queries that might be useful to the learner. The purpose of this section is simply to point out that the construction of Section 2-3 is applicable even in the presence of most kinds of queries. That is, a weak learning algorithm that depends on the availability of certain kinds of queries can be converted, using the same construction, into a strong learning algorithm using the same query types.

2-5.5   Many-valued concepts

In this chapter, we have only considered Boolean-valued concepts, i.e., concepts that classify every instance as either a positive or a negative example. Of course, in the "real world," many learning tasks require classification into one of several categories (for instance, character recognition). How does the result generalize to handle many-valued concepts?

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

Thus, a better definition of weak learnability is one requiring that the hypothesis be correct on slightly more than half of the distribution, regardless of k. Using this definition, the construction of Section 2-3 is easily modified to handle many-valued concepts.

2-6   General complexity bounds for PAC learning

The construction derived in Sections 2-3 and 2-4 yields some unexpected relationships between the allowed error ϵ and various complexity measures that might be applied to a strong learning algorithm. One of the more surprising of these is a proof that, for every learnable concept class,

there exists an efficient algorithm whose output hypotheses can be evaluated in time polynomial in $\log(1/\epsilon)$. Furthermore, such an algorithm's space requirements are also only poly-logarithmic in $1/\epsilon$—far less, for instance, than would be needed to store the entire sample. In addition, its time and sample size requirements grow only linearly in $1/\epsilon$ (disregarding log factors).

Theorem 6.1 *If C is a learnable concept class, then there exists an efficient learning algorithm for C that:*

- *requires a sample of size $\dfrac{1}{\epsilon} \cdot p_1(n, s, \log(1/\epsilon), \log(1/\delta))$,*

- *halts in time $\dfrac{1}{\epsilon} \cdot p_2(n, s, \log(1/\epsilon), \log(1/\delta))$,*

- *uses space $p_3(n, s, \log(1/\epsilon), \log(1/\delta))$, and*

- *outputs hypotheses of size $p_4(n, s, \log(1/\epsilon))$, evaluatable in time $p_5(n, s, \log(1/\epsilon))$.*

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

Proof: Given a strong learning algorithm A for C, "hard-wire" $\epsilon = 1/4$, thus converting A into a weak learning algorithm A' that outputs hypotheses $1/4$-close to the target concept. Now let A'' be the procedure obtained by applying the construction of **Learn'** with A' plugged in for **WeakLearn**. As remarked previously, we can assume without loss of generality that A'' halts deterministically in polynomial time. Note, by the lemmas of Sections 2-3 and 2-4 that A'' "almost" achieves the resource bounds given in the theorem, the only problem being that the bounds attained are polynomial in $1/\delta$ rather than $\log(1/\delta)$ as desired.

This problem is alleviated by applying the construction of Haussler et al. [38] for converting any learning algorithm B into one running in time polynomial in $\log(1/\delta)$. Essentially, this construction works as follows: Given inputs n, s, ϵ and δ, first simulate B $O(\log(1/\delta))$ times, each time setting B's accuracy parameter to $\epsilon/4$ and B's confidence parameter to $1/2$. Save all of the computed hypotheses. Next, draw a sample of $O(\log(1/\delta)/\epsilon)$ examples, and output the one that misclassifies the fewest examples in the sample. Haussler et al. argue that the resulting procedure outputs an ϵ-close hypothesis with probability $1 - \delta$.

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

The remainder of this section is a discussion of some of the consequences of Theorem 6.1.

2-6.1   Improving the performance of existing algorithms

These bounds can be applied immediately to a number of existing learning algorithms, yielding improvements in time and/or space complexity (at least in terms of ϵ). For instance, the computation time of Blumer et al.'s algorithm [14] for learning half-spaces of $\mathbb{R}^n$, which involves the

solution of a linear programming problem of size proportional to the sample, can be improved by a polynomial factor of $1/\epsilon$. The same is also true of Baum's [11] algorithm for learning unions of half-spaces, which involves finding the convex hull of a significant fraction of the sample.

There are many more algorithms for which the theorem implies improved space efficiency. This is especially true of the many known PAC algorithms that work by choosing a large sample and then finding a hypothesis consistent with it. For instance, this is how Rivest's [72] decision list algorithm works, as do most of the algorithms described by Blumer et al. [14], as well as Helmbold, Sloan and Warmuth's [43] construction for learning nested differences of learnable concepts. Since the entire sample must be stored, these algorithms are not terribly space efficient, and so can be dramatically improved by applying Theorem 6.1. Of course, these improvements typically come at the cost of requiring a somewhat larger sample (by a polynomial factor of $\log(1/\epsilon)$). Thus, there appears to be a trade-off between sample size and space (or time) complexity.

2-6.2   Data compression

Blumer et al. [13, 14] have considered the relationship between learning and data compression. They have shown that, if any sample can be "compressed"—i.e., represented by a prediction rule significantly smaller than the original sample—then this compression algorithm can be converted into a PAC-learning algorithm.

In some sense, the bound given in Theorem 6.1 on the size of the output hypothesis implies the converse. In particular, suppose C is a learnable concept class and that we have been given m examples $(x_1, c(x_1)), (x_2, c(x_2)), \ldots, (x_m, c(x_m))$ where each $x_i \in X_n$ and c is a concept in C_n of size s. These examples need not have been chosen at random. The data compression problem is to find a small representation for the data, i.e., a hypothesis h that is significantly smaller than the original data set with the property that $h(x_i) = c(x_i)$ for each x_i. A hypothesis with this last property is said to be *consistent* with the sample.

Theorem 6.1 implies the existence of an efficient algorithm that outputs consistent hypotheses only poly-logarithmic in the size m of the sample. This is proved by the following theorem:

Theorem 6.2 *Let C be a learnable concept class. Then there exists an efficient algorithm that, given $0 < \delta \leq 1$ and m (distinct) examples of a concept $c \in C_n$ of size s, outputs with probability at least $1 - \delta$ a deterministic hypothesis consistent with the sample of size polynomial in n, s and $\log m$.*

Proof: Pitt and Valiant [68] show how to convert any learning algorithm into one that finds hypotheses consistent with a set of data points. The idea is to choose $\epsilon < 1/m$ and to run the learning algorithm on a (simulated) uniform distribution over the data set. Since ϵ is less than

the weight placed on any element of the sample, the output hypothesis cannot misclassify even a single data point. Applying this technique to a learning algorithm A satisfying the conditions of Theorem 6.1, we see that the output hypothesis has size only polynomial in n, s and $\log m$, and so is far smaller than the original sample for large m.

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

Discrete domains

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

Corollary 6.3 *Let C be a learnable discrete concept class. Then there exists an efficient algorithm that, given $0 < \delta \leq 1$ and a sample as in Theorem 6.2, outputs with probability at least $1 - \delta$ a deterministic consistent hypothesis of size polynomial in n and s, and independent of m.*

Proof: Since we assume (without loss of generality) that all the points of the sample are distinct, the sample size m cannot exceed $|X_n|$. Since $\log |X_n|$ is bounded by a polynomial in n, the corollary follows immediately.   ■

Applying "Occam's Razor" of Blumer et al. [13], this implies the following strong general bound on the sample size needed to efficiently learn C. Although the bound is better than that

given by Theorem 6.1 (at least in terms of ϵ), it should be pointed out that this improvement requires the sacrifice of space efficiency since the entire sample must be stored.

Theorem 6.4 *Let C be a learnable discrete concept class. Then there exists an efficient learning algorithm for C requiring a sample of size $O\left(\epsilon^{-1} \cdot (p(n, s) + \log(1/\delta))\right)$ for some polynomial p.*

Proof: Blumer et al. [13] describe a technique for converting a so-called "Occam" algorithm A with the property described in Corollary 6.3 into an efficient learning algorithm with the stated sample complexity bound. Essentially, to make this conversion, one simply draws a sample of the stated size (choosing p appropriately), and runs A on the sample to find a consistent hypothesis. The authors argue that the computed hypothesis, simply by virtue of its small size and consistency with the sample, will be ϵ-close to the target concept with high probability. (Technically, their approach needs some minor modifications to handle, for instance, a randomized Occam algorithm; these modifications are straightforward.) ∎

Non-discrete domains

Littlestone and Warmuth [59] consider the relationship between learning and more general kinds of data compression schemes applicable to domains that are not necessarily discrete. Specifically, a *compression scheme* is an algorithm that takes as input a sample S of m labeled examples of some target concept c, and that outputs a hypothesis h consistent with the sample S and represented over the alphabet S. In other words, h is represented by a sequence of examples from the sample itself. For example, if the domain is the real plane, and the hypothesis classifies points as positive if and only if they occur inside some rectangle determined by four points from the sample, then h is naturally represented by those four examples.

The *kernel size* of the hypothesis is just the length of the sequence of examples that represents it. Also, it is often convenient to allow the hypothesis to incorporate "additional information" — say, a sequence of bits providing some supplementary information about the hypothesis. The size of the hypothesis is then its total length in symbols over the alphabet $S \cup \{0, 1\}$; that is, its length is equal to its kernel size plus the length of any additional information. As usual, we require that the compression algorithm A run in polynomial time, and that the output hypothesis be polynomially evaluatable.

Littlestone and Warmuth [59] show that if there exists a compression scheme algorithm A for a concept class C that outputs hypotheses significantly smaller than the sample (say, linear in m^α for some constant $\alpha < 1$) then A can be used as a learning algorithm and C is learnable.

As a consequence of Theorem 6.2. we can show that the converse holds as well: if C is learnable, then there exists a compression scheme for C outputting hypotheses of size polynomial in n, s and $\log m$.

Theorem 6.5 *Let C be a learnable concept class. Then there exists a compression scheme algorithm for C that, given a sample S of m examples of some size-s concept $c \in C_n$, outputs a hypothesis h consistent with S, and represented as a string in $\{0,1\}^\ell \times S^k$ for some k and ℓ polynomial in n, s and $\log m$.*

Proof: Let A be a weak learning algorithm for C. The key point in this proof is that hypotheses output by A can be trivially represented by the entire set of examples actually received by the algorithm: a hypothesis represented in this manner can be efficiently evaluated by re-running A on the sample (given by the hypothesis) producing the "true" hypothesis of A which can then be evaluated on a given instance. (If A is randomized, we also include the random bits that were used.) Note that such a hypothesis has kernel size equal to the (polynomial) sample complexity of A.

Next, we apply the hypothesis-boosting construction of Sections 2-3 and 2-4, and we then eliminate any dependence on δ in the size of the output hypothesis using the technique described in the proof of Theorem 6.1. The result is a strong learning algorithm for C that outputs hypotheses with kernel size only polynomial in n, s and $\log(1/\epsilon)$. Specifically, the hypothesis is represented by the examples used on each simulated execution of A, in addition to a bit string describing the overall structure of the hypothesis constructed by the boosting procedure. Finally, this hypothesis (which may be randomized) is converted into a compression scheme as described in Theorem 6.2.   ■

Thus, a necessary and sufficient condition for learnability is the existence of a compression scheme of the style described by Littlestone and Warmuth [59].

It is worth pointing out that the technique described in Theorem 6.5, combined with the results of Littlestone and Warmuth, gives an alternative method for analyzing the sample complexity of the hypothesis boosting procedure of Section 2-3. Specifically, this proof shows that **Learn** can be used as a compression scheme with output hypothesis size polynomial in n, s and $\log m$. Littlestone and Warmuth show that such a compression scheme can in turn be converted into a learning algorithm with sample size $\epsilon^{-1} \cdot p(n, s, \log(1/\epsilon), \log(1/\delta))$ for some polynomial p.

Furthermore, notice that when **Learn** is used as a data compression scheme (using the technique described in Theorem 6.5) the target distribution is entirely known, both at the top level where it is uniform on the sample, and (with some simple modifications) at each lower level as well. Thus, in such a case, there is no need to hypothesis test since the error of any hypothesis can be computed directly by evaluating it on each instance and using our knowledge about the target distribution. Also, there is no longer a need to filter any distributions — the given distribution can be directly simulated. For these reasons, the algorithm can be shown to run much faster — in fact, its running time is comparable to the sample size stated above.

Thus, Littlestone and Warmuth's techniques can be used to modify Learn, yielding time and sample complexity bounds that, like those of Learn', are linear in $1/\epsilon$ (ignoring log factors). However, in contrast to Learn', the resulting procedure is *not* space efficient since the entire sample must be stored.

2-6.3   Hard functions are hard to learn

Theorem 6.1's bound on the size of the output hypothesis also implies that any hard-to-evaluate concept class is unlearnable. Although this result does not sound surprising, it was previously unclear how it might be proved: since a learning algorithm's hypotheses are technically permitted to grow polynomially in $1/\epsilon$, the learnability of such classes did not seem out of the question.

This result yields the first representation-independent hardness results not based on cryptographic assumptions. For instance, assuming $P/poly \neq NP/poly$, the class of polynomial-size, nondeterministic Boolean circuits is not learnable. (The set $P/poly$ ($NP/poly$) consists of those languages accepted by a family of polynomial-size deterministic (nondeterministic) circuits.) Furthermore, since learning pattern languages was recently shown [77] to be as hard as learning $NP/poly$, this result shows that pattern languages are also unlearnable under this relatively weak complexity-theoretic assumption.

Theorem 6.6 *Suppose C is learnable, and assume that $X_n = \{0,1\}^n$. Then there exists a polynomial p such that for all concepts $c \in C_n$ of size s, there exists a circuit of size $p(n,s)$ exactly computing c.*

Proof: Consider the set of 2^n pairs $\{(x, c(x)) : x \in X_n\}$. By Corollary 6.3, there exists an algorithm that, with positive probability, will output a hypothesis consistent with this set of elements of size only polynomial in n and s. Since this hypothesis is polynomially evaluatable, it can be converted using standard techniques into a circuit of the required size. ∎

2-6.4   Hard functions are hard to approximate

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

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

Theorem 6.7 *Suppose some function f cannot be computed by any family of polynomial-size circuits. Then there exists a family of distributions $D_1, D_2, \ldots$, where D_n is over the set $\{0,1\}^n$,*

such that for all polynomials p and q, there exist infinitely many n for which there exists no n-variable circuit of size at most $q(n)$ that $\left(\frac{1}{2} - \frac{1}{p(n)}\right)$-approximates f under D_n.

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

Suppose first that there exists some k such that for all n and every distribution D on $\{0,1\}^n$, there exists a circuit of size at most n^k that $\left(\frac{1}{2} - \frac{1}{n^k}\right)$-approximates f under D. Then f can, in a sense, be weakly learned. More precisely, there exists an (exponential-time) procedure that, by searching exhaustively the set of all circuits of size n^k, will find one that $\left(\frac{1}{2} - \frac{1}{n^k}\right)$-approximates f under some given distribution D. Therefore, by Theorem 3.1, f is strongly learnable in a similar sense in exponential time. Applying Theorem 6.6 (whose validity depends only on the size of the output hypothesis, and not on the running time), this implies that f can be exactly computed by a family of polynomial-size circuits, contradicting the theorem's hypothesis.

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

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

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

Informally, Theorem 6.7 states that any language not in the complexity class P/poly cannot be even weakly approximated by any other language in P/poly under some "hard" family of distributions. In fact, the theorem can easily be modified to apply to other circuit classes as well, including monotone P/poly, and monotone or non-monotone NC^k for fixed k. (The class NC^k consists of all languages accepted by polynomial-size circuits of depth at most $O(\log^k n)$, and a monotone circuit is one in which no negated variables appear.) In general the theorem applies to all circuit classes closed under the transformation on hypotheses resulting from the construction of Sections 2-3 and 2-4.

2-6.5   On-line learning

Finally, we consider implications of Theorem 6.1 for on-line learning algorithms. In the *on-line* learning model, the learner is presented one (randomly selected) instance at a time in a series of *trials*. Before being told its correct classification, the learner must try to predict whether the instance is a positive or negative example. An incorrect prediction is called a *mistake*. In this model, the learner's goal is to minimize the number of mistakes.

Previously, Haussler, Littlestone and Warmuth [39] have shown that a concept class C is learnable if and only if there exists an on-line learning algorithm for C with the properties that:

- the probability of a mistake on the mth trial is at worst linear in $m^{-\beta}$ for some constant $0 < \beta \le 1$, and (equivalently)

- the expected number of mistakes on the first m trials is at worst linear in m^{α} for some constant $0 \le \alpha < 1$.

(This result is also described in their paper with Kearns [38].) Noting several examples of learning algorithms for which this second bound only grows poly-logarithmically in m, the authors ask if *every* learnable concept class has an algorithm attaining such a bound. Theorem 6.8 below answers this open question affirmatively, showing that in general the expected number of mistakes on the first m trials need only grow as a polynomial in $\log m$. Thus, we expect only a minute fraction of the first m predictions to be incorrect.

(This result should not be confused with those presented in another paper by Haussler, Littlestone and Warmuth [40]. In this chapter, the authors describe a general algorithm applicable to a wide collection of concept classes, and they show that the expected number of mistakes made by this algorithm on the first m trials is linear in $\log m$. However, their algorithm requires exponential computation time, even if it is known that the concept class is learnable. In contrast, Theorem 6.8 states that, if a concept class is learnable, then there exists an efficient algorithm making poly-logarithmic in m mistakes on average on the first m trials.)

Haussler, Littlestone and Warmuth [39] also consider the space efficiency of on-line learning algorithms. They define a *space-efficient* learning algorithm to be one whose space requirements on the first m trials do not exceed a polynomial in n, s and $\log m$. Thus, a space efficient algorithm is one using far less memory than would be required to store explicitly all of the preceding observations. The authors describe a number of space-efficient algorithms (though are unable to find one for learning unions of axis-parallel rectangles in the plane), and so are led to ask whether there exist space-efficient algorithms for *all* learnable concept classes. Surprisingly, this open question can also be answered affirmatively, as proved by the theorem below.

Lastly, Theorem 6.8 gives a bound on the computational complexity of on-line learning (in terms of m). In particular, the total computation time required to process the first m examples is only proportional to $m \log^c m$, for some constant c. Thus, in a sense, the "amortized" or "average" computation time on the mth trial is only poly-logarithmic in m. (In fact, a more careful analysis would show that this is also true of the worst-case computation time on the mth trial.)

Theorem 6.8 *Let C be a learnable concept class. Then there exists an efficient on-line learning algorithm for C with the properties that:*

- *the probability of a mistake on the mth trial is at most $m^{-1} \cdot p_1(n, s, \log m)$,*

- *the expected number of mistakes on the first m trials is at most $p_2(n, s, \log m)$,*

- *the total computation time required on the first m trials is at most $m \cdot p_3(n, s, \log m)$, and*

- *the space used on the first m trials is at most $p_4(n, s, \log m)$,*

for some polynomials p_1, p_2, p_3, p_4.

Proof: Since C is learnable, there exists an efficient (batch) algorithm satisfying the properties of Theorem 6.1. Let A be such an algorithm, but with $\epsilon/2$ substituted for both ϵ and δ. Then the chance that A's output hypothesis incorrectly classifies a randomly chosen instance is at most ϵ. (This technique is also used by Haussler et al. [38].)

Fix n and s, and let $m(\epsilon)$ be the number of examples needed by A. From Theorem 6.1, $m(\epsilon) \leq (p/\epsilon) \cdot \lg^c(1/\epsilon)$ for some constant c and some value p implicitly bounded by a polynomial in n and s. Let $\epsilon(m) = (p/m) \cdot \lg^c(m/p)$. Then it can be verified that $m(\epsilon(m)) \leq m$ for $m \geq 2p$. Thus, m examples suffice to find a hypothesis whose chance of error is at most $\epsilon(m)$.

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

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

Thus, if the mth trial occurs in the ith block (i.e., if $a_i \leq m \leq b_i$), then the probability of a mistake is bounded by $\epsilon(s_{i-1})$, the error rate of h_{i-1}. From the definition of $\epsilon()$, this implies the

desired bound on the probability of a mistake on the mth trial, and, in turn, on the expected number of mistakes on the first m trials.

Finally, note that on the ith block, space is needed only to store the hypothesis from the last block h_{i-1}, and to simulate A's computation of block i's hypothesis. By Theorem 6.1, both of these quantities grow polynomially in $\log(1/\epsilon)$. By our choice of ϵ, this implies the desired bound on the algorithm's space efficiency. The time complexity of the procedure is bounded in a similar fashion. ∎

2-7   Conclusions and open problems

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

It is hoped that these results will open the way on a new method of algorithm design for PAC-learning. As previously mentioned, the vast majority of currently known algorithms work by finding a hypothesis consistent with a large sample. An alternative approach suggested by the main result is to seek instead a hypothesis that works correctly on slightly more than half the distribution. Perhaps, such a hypothesis is easier to find, at least from the point of view of the algorithm designer. This approach leads to algorithms with a flavor similar to the one described for k-term DNF in Section 2-5.3. To what extent will this approach be fruitful for other classes not presently known to be learnable?

Another open question concerns the robustness of the construction described in this chapter. Intuitively, it seems that there should be a close relationship between reducing the error of the hypothesis, and overcoming noise in the data. Is this a valid intuition? Can our construction be modified to handle noise? Can the construction be extended to the p-concept model described in Chapter 4?

Finally, turning away from the theoretical side of machine learning, we can ask how well our construction would perform in practice. Often, a learning program (for instance, a neural network) is designed, implemented, and found empirically to achieve a "good" error rate, but no way is seen of improving the program further to enable it to achieve a "great" error rate. Suppose our construction is implemented on top of this learning program. Would it help? This is not a theoretical question, but one that can only be answered experimentally, and one that obviously depends on the domain and the underlying learning program. Nevertheless, it seems plausible that the construction might in some cases give good results in practice.

Statistical-perturbation Methods for Inference of Read-once Formulas

3-1 Introduction

This chapter explores in detail a simple but powerful statistical technique for discovering the structure of a read-once formula. (A formula is *read-once* if each variable appears at most once in the formula.) As a demonstration of its power, we apply this technique to an array of learning problems; in each case, we obtain the first provably efficient algorithm that effectively solves the given learning problem. We also demonstrate that our method is highly robust against a great deal of noise and randomness.

Similar to the Valiant model [83], we consider the problem of learning read-once formulas from randomly chosen examples. The basic idea of our method is to observe the statistical behavior of the target formula's output under various simple and easily sampled perturbations of the target distribution (the distribution under which random examples are chosen). For example, a typical perturbation might "hard-wire" a single variable to some fixed value. In a variety of situations, we demonstrate that this simple technique can be applied to effectively discover much or all of the target formula's structure.

For example, using this method, we are able to derive efficient algorithms for *exactly iden-tifying* certain classes of read-once Boolean formulas when the observed examples are chosen randomly according to specific, fixed and simple distributions. Even when the formula's output is corrupted by a great deal of random misclassification noise, we show that exact identification can be achieved.

We also apply our method to a probabilistic generalization of the class of all read-once Boolean formulas constructed from the usual basis {AND, OR, NOT}. We show that an arbitrarily

good approximation of such formulas can be inferred in polynomial time against any *product distribution* (i.e., any distribution in which the setting of each variable is chosen independently of the settings of the other variables). For example, this shows that the class of read-once Boolean formulas over the usual basis can be learned in polynomial time against the uniform distribution in the sense of Valiant.

The problem of learning Boolean formulas against special distributions has been considered by a number of other authors. In particular, our technique closely resembles that used by Kearns et al. [51] for learning the class of read-once formulas in disjunctive normal form (DNF) against the uniform distribution. A similar result, though based on a different method, was obtained by Pagallo and Haussler [66]. Our results extend theirs to a much broader class of read-once formulas.

Also, Linial, Mansour and Nisan [57] used a technique based on Fourier spectra to learn the class of constant-depth formulas (constructed from gates of unbounded fan-in) against the uniform distribution. Furst, Jackson and Smith [27] generalized this result to learn this same class against any product distribution. Verbeurgt [86] gives a different algorithm for learning DNF-formulas against the uniform distribution. However, all three of these algorithms require quasi-polynomial ($n^{\text{polylog}(n)}$) time, though Verbeurgt's procedure only requires a polynomial-size sample.

Exact identification using amplification functions

As mentioned above, this chapter includes efficient algorithms for *exactly identifying* certain classes of read-once Boolean formulas by observing the target formula's behavior on examples drawn randomly according to a fixed and simple distribution. This distribution is related to the formula's *amplification function*. The amplification function $A_f(p)$ for a function $f : \{0,1\}^n \to \{0,1\}$ is defined as the probability that the output of f is 1 when each of the n inputs to f is 1 independently with probability p. Amplification functions were first studied by Valiant [83] and Boppana [16, 17] in obtaining bounds on monotone formula size for the majority function.

The method used by our algorithms is of central interest. For several classes of formulas, we show that the behavior of the amplification function is *unstable* near the fixed point; that is, the value of $A_f(p)$ varies greatly with a small change in p. This in turn implies that small but easily sampled perturbations of the fixed-point distribution (that is, the distribution where each input is 1 with probability p, where $A_f(p) = p$) reveal structural information about the formula. As mentioned above, a typical perturbation of the fixed-point distribution hard-wires a single variable to 1 and sets the remaining variables to 1 with probability p.

We apply this method to obtain efficient algorithms for exact identification of classes of read-once formulas over various bases. These include the class of logarithmic-depth read-once

formulas constructed with NOT gates and three-input majority gates (for which the fixed-point distribution is the uniform distribution), as well as the class of logarithmic-depth read-once formulas constructed with NAND gates (for which the fixed-point distribution assigns 1 to each input independently with probability $1/\phi \approx 0.618$, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio). Thus, for these classes, since the fixed point of the amplification function is the same for all formulas, we obtain a single simple distribution for the entire class. As proved by Kearns and Valiant [52, 49], these same classes of formulas cannot be even weakly approximated in polynomial time when no restriction is placed on the target distribution; thus, our results may be interpreted as demonstrating that while there are some distributions which in a computationally bounded setting reveal essentially *no* information about the target formula, there are natural and simple distributions which reveal *all* information.

For Boolean read-once formulas (a superset of the class of formulas constructed from NAND gates) there is an efficient, exact-identification algorithm using membership and equivalence queries due to Angluin, Hellerstein and Karpinski [8, 41]. The class of read-once majority formulas can also be exactly identified using membership and equivalence queries, as proved by Hancock [33] and Hellerstein and Karpinski [42]. Briefly, in the query model, the learner attempts to infer the target formula by asking question, or *queries*, of a "teacher." For instance, the learner might ask the teacher what the formula's output would be for a specific assignment to the input variable; this is called a *membership query*. On an *equivalence query*, the learner asks if a given conjectured formula is equivalent to the target formula

Note that our algorithms' use of a *fixed* distribution can be regarded as a form of "random" membership queries, since this fixed and known distribution can be easily simulated by making random membership queries. Thus, our algorithms are the first efficient procedures for exact identification of logarithmic-depth majority and NAND formulas using only membership queries. Furthermore, the queries used are *non-adaptive* in the sense that they do not depend upon the answers received to previous queries. In contrast, all previous algorithms for exact identification, including the algorithms mentioned above, require highly adaptive queries.

We also prove that our algorithms are *robust* against a large amount of *random misclassification noise*, similar to, but slightly more general than that considered by Sloan [81] and Angluin and Laird [9]. Specifically, if η_0 and η_1 represent the respective probabilities that an output of 0 or 1 is misclassified, then a robust version of our algorithm can handle any noise rate for which $\eta_0 + \eta_1 \neq 1$; the sample size and computation time required increase only by an inverse quadratic factor in $|1 - \eta_0 - \eta_1|$. Again regarding our algorithms as using "random" membership queries, these are the first efficient procedures performing exact identification in some reasonable model of *noisy queries*. Our algorithms can also tolerate a modest rate of *malicious noise*.

Finally, we present an algorithm that learns *any* (not necessarily logarithmic-depth) read-once majority formula in Valiant's model against the uniform distribution. To obtain this result we first show that the target formula can be well approximated by truncating the formula to have only logarithmic depth. We then generalize our algorithm for learning logarithmic-depth read-once formulas to handle such truncated formulas. A similar result also holds for read-once NAND formulas of unbounded depth.

Probabilistic read-once formulas

In Section 3-7, we describe an algorithm for learning a probabilistic generalization of the class of read-once formulas over the usual basis {AND, OR, NOT}.

We adopt from Chapter 4 the notion of a *probabilistic concept (p-concept)*. A p-concept c is a function which maps each input-variable assignment x to a real number $c(x)$ between 0 and 1. We interpret $c(x)$ as the *probability* that instance x will be positively classified. Thus, in the p-concept model, a randomly labeled example is chosen as follows: first, an instance x is chosen at random according to the target distribution on the instance space; then, with probability $c(x)$, the labeled example $(x, 1)$ is observed, and with probability $1 - c(x)$, the labeled example $(x, 0)$ is observed. Thus, in general, the learner has no direct access to the function c, even on individual points.

We view the learning problem as that of inferring from such randomly chosen examples a good approximation of the function c itself. Thus, we ask that the learner infer a real-valued hypothesis h for which $|h(x) - c(x)|$ is small for most instances x. This is called *learning with a model of probability*.

Specifically, we consider the problem of learning a class of real-valued read-once formulas, called *read-once real formulas*. Formulas in this class are constructed using two kinds of gates, or operators: The first gate, denoted MUL, simply multiplies its two real-valued inputs. The second gate, LIN_{zw}, computes the function $\text{LIN}_{zw}(y) = z + wy$. Here, z and w may be any real numbers for which z and $z + w$ are both in the range $[0, 1]$. Clearly, for a Boolean assignment to the input variables, a formula constructed from such gates outputs a real number between 0 and 1, and so these are indeed p-concepts. We show that this class can be learned with a model of probability against any product distribution (such as the uniform distribution).

Note that, for Boolean-valued inputs, the function MUL simply computes the logical AND of its inputs, and $\text{LIN}_{1,-1}$ computes the logical negation of its input. Thus, the class of read-once real formulas includes the class of read-once Boolean formulas with basis {AND, OR, NOT}. Therefore, our result demonstrates for the first time the existence of a polynomial-time algorithm for inferring a good approximation of any such Boolean formula (against a product distribution).

Also, a gate LIN_{zw} can alternatively be viewed as describing the behavior of a noisy or random Boolean gate which, on input 0 randomly outputs 1 with probability z, and on input 1 outputs 1 with probability $z+w$. (If the input to such a randomized gate is 1 with probability p, then the output is easily computed to be 1 with probability $\text{LIN}_{zw} = z + wp$.) Thus, for the distributions considered, our result can be regarded as a demonstration of the learnability of read-once Boolean formulas, even when every gate and every wire of the formula is corrupted by significant amounts of randomness.

3-2   Preliminaries

Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, Boppana [16, 17] defines its *amplification function* A_f as follows: $A_f(p) = \Pr[f(X_1,\ldots,X_n) = 1]$, where $X_1,\ldots,X_n$ are independent Bernoulli variables that are each 1 with probability p. The quantity $A_f(p)$ is called the *amplification of f at p*. Valiant [83] uses properties of the amplification function to prove the existence of monotone Boolean formulas of size $\mathcal{O}(n^{5.3})$ for the majority function on n inputs. Also, we denote by $D^{(p)}$ the distribution over $\{0,1\}^n$ induced by having each variable independently set to 1 with probability p.

For $q_j \in \{0,1\}$ and $i_j \in \{1,\ldots,n\}$, $1 \leq j \leq r$, we write $f|x_{i_1} \leftarrow q_1,\ldots,x_{i_r} \leftarrow q_r$ to denote the function obtained from f by fixing or hard-wiring each variable x_{i_j} to the value q_j. If each $q_j = q$ for some value q, we abbreviate this by $f|x_{i_1},\ldots,x_{i_r} \leftarrow q$.

In our framework, the learner is attempting to infer an unknown *target concept* c chosen from some known *concept class* C. In this chapter, $C = \bigcup_{n \geq 1} C_n$ is parameterized by the number of variables n, and each $c \in C_n$ represents a Boolean function on the domain $\{0,1\}^n$. A polynomial-time learning algorithm achieves *exact identification* of a concept class (from some source of information about the target, such as examples or queries) if it can infer a concept that is equal to the target concept on all inputs. A polynomial-time learning algorithm achieves *exact identification with high probability* if for any $\delta > 0$, it can with probability at least $1 - \delta$ infer a concept that is equal to the target concept on all inputs. In this setting polynomial time means polynomial in n and $1/\delta$. Our algorithms achieve exact identification with high probability when the example source is a particular, fixed distribution.

In the *distribution-free* or *probably approximately correct* (PAC) learning model, introduced by Valiant [83] and described in previous chapters, the learner is given access to labeled (positive and negative) examples of the target concept, drawn randomly according to some unknown *target distribution* D. The learner is also given as input $\epsilon, \delta > 0$. The learner's goal is to output with probability at least $1 - \delta$ a *hypothesis* h that has probability at most ϵ of disagreeing with c on a randomly drawn example from D (thus, the hypothesis has *accuracy* at least $1 - \epsilon$). If such a learning algorithm exists (that is, a polynomial-time algorithm meeting the goal for any

$n \geq 1$, any $c \in \mathcal{C}_n$, any distribution D, and any ϵ, δ), we say that $\mathcal{C}$ is *PAC-learnable*. In this setting, polynomial time means polynomial in n, $1/\epsilon$ and $1/\delta$. In this chapter, we will primarily be interested in a variant of Valiant's model in which the target distribution is known a priori to belong to a specific restricted class of distributions.

Note that because we consider only read-once formulas, there is a unique path from any gate or variable to the output. We define the *level* or *depth of a gate* λ to be the number of gates (not including λ itself) on the path from λ to the output. Thus, the output gate is at level 0. Likewise, we define the *level* or *depth of an input variable* to be the number of gates on the path from the variable to the output. The *depth* of the entire formula is the maximum level of any input, and the *bottom level* consists of all gates and variables of maximum depth.

An input x_i, or a gate λ, *feeds* a gate λ' if the path from x_i or λ to the output goes through λ'. If x_i or λ is an input to λ', then we say that x_i or λ *immediately feeds* λ'. For any two input bits x_i and x_j we define $\Gamma(x_i, x_j)$ to be the deepest gate λ fed by both x_i and x_j. Likewise, $\Gamma(x_i, x_j, x_k)$ is the deepest gate λ fed by x_i, x_j, and x_k. We say that a pair of variables x_i and x_j *meet* at the gate $\Gamma(x_i, x_j)$. Also, if $\Gamma(x_i, x_j) = \Gamma(x_i, x_k) = \Gamma(x_j, x_k) = \Gamma(x_i, x_j, x_k)$, then we say that the variables x_i, x_j and x_k *meet* at gate $\Gamma(x_i, x_j, x_k)$; otherwise, the triple does not meet in the formula. (Note that this only makes sense if there are gates with more than two inputs, such as a three-input majority gate.)

3-3   Exact identification of read-once majority formulas

In this section we use properties of amplification functions to obtain a polynomial-time algorithm that with high probability exactly identifies any read-once majority formula of logarithmic depth from random examples drawn according to a uniform distribution.

This type of formula is used in Chapter 2's proof that a concept class is weakly learnable in polynomial time if and only if it is strongly learnable in polynomial time. That is, the hypothesis output by the boosting procedure described in that chapter can be viewed as a majority formula whose inputs are the hypotheses output by the weak learning algorithm. We also note that a read-once majority formula cannot in general be converted into a read-once Boolean formula over the usual {AND, OR, NOT} basis.

It can be shown that the class of logarithmic-depth read-once majority formulas is not learnable in the distribution-free model (modulo some cryptographic assumptions; see Kearns and Valiant [52] for details). Briefly, this can be proved using a Pitt and Warmuth-style "prediction-preserving reduction" [70] to show that learning read-once majority formulas is at least as hard as learning general Boolean formulas. Given a Boolean formula (of logarithmic depth, without loss of generality), the main idea of such a prediction-preserving reduction is to replace each OR gate (respectively, AND gate) with a MAJ gate, one of whose inputs is

wired to a variable that, under the target distribution, always has the value 1 (respectively, 0). The resulting majority formula can further be reduced to one that is read-once using the substitution method of Kearns et al. [51]. Finally, combined with Kearns and Valiant's result that Boolean formulas are not learnable (modulo cryptographic assumptions), this shows that majority formulas are also unlearnable.

Despite the hardness of this class in the general distribution-free framework, we show that the class is nevertheless exactly identifiable when examples are chosen from the uniform distribution. The algorithm consists of two phases. In the first phase, we determine the relevant variables (i.e., those that occur in the formula), their signs (i.e., whether they are negated or not), and their levels. To achieve this goal, for each variable, we hard-wire its value to 1 and estimate the amplification of the induced function at $\frac{1}{2}$ using examples drawn randomly from the uniform distribution on the remaining variables. Here, by "hard-wiring" a variable to 1, we really mean that we apply a *filter* that only lets through examples for which that variable is 1. We prove that if the variable is relevant, then with high probability this estimate will be significantly smaller or greater than $\frac{1}{2}$, depending on whether the variable occurs negated or unnegated in the formula; otherwise, this estimate will be near $\frac{1}{2}$. Furthermore, the level of a relevant variable can be determined from the amount by which the amplification of the induced function differs from $\frac{1}{2}$.

In the second phase of the algorithm, we construct the formula. More precisely, we first construct the bottom level of the formula, and then recursively construct the remaining levels. To construct the bottom level of the formula, we begin by finding triples of variables that are inputs to the same bottom-level gate. To do this, for each triple of relevant variables that have the largest level number, we hard-wire the three variables to 1 and again estimate the amplification of the induced function from random examples. We show that we can determine whether the three variables all enter the same bottom-level gate based on this estimate.

Briefly, the recursion works as follows. Suppose that we are currently constructing level t of the formula and we find that x_i, x_j, and x_k are inputs to the same level-t gate. Then in the recursive call we replace x_i, x_j, and x_k by a level-t meta-variable $y \equiv \text{MAJ}(x_i, x_j, x_k)$. Since y is a known subformula, its output on any example can be easily computed and y can be treated like an ordinary variable. Furthermore, since $\frac{1}{2}$ is the fixed point for the amplification function of any read-once majority formula, it follows that y is 1 with probability $\frac{1}{2}$. Thus, for the recursive call we replace all triples of variables that enter level-t gates with meta-variables, and we easily obtain our needed source of random examples drawn according to the uniform distribution on the new variable set from the original source of examples.

For the remainder of this section, we explore some of the properties of the amplification function of read-once majority formulas, leading eventually to a proof of the correctness of this
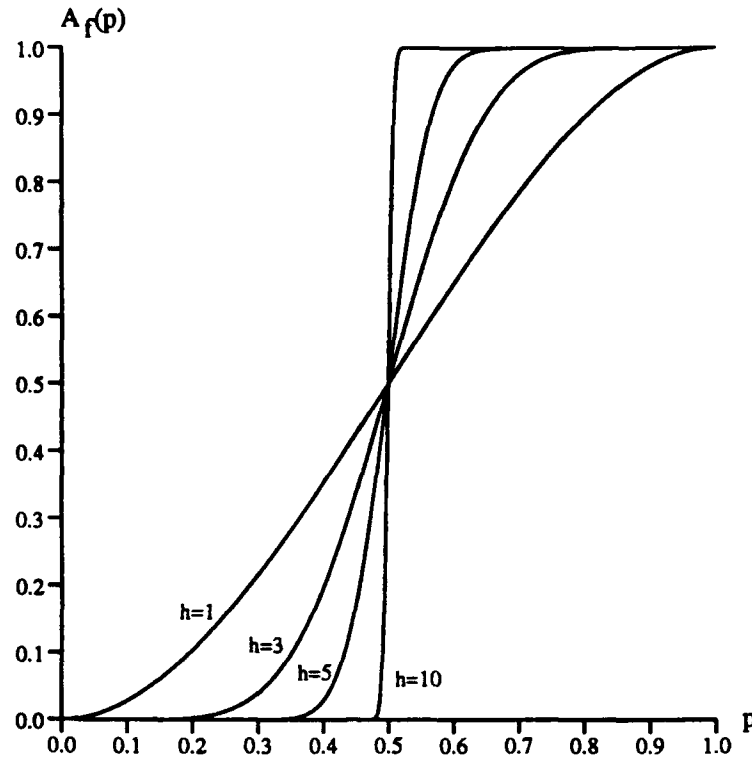
Figure 1: The amplification function for read-once majority formulas for complete ternary trees of depth h.

algorithm.

Lemma 3.1 *Let X_1, X_2 and X_3 be three independent Bernoulli variables, each 1 with probability p_1, p_2 and p_3, respectively. Then* $\Pr[\mathrm{MAJ}(X_1, X_2, X_3) = 1] = p_1 p_2 + p_1 p_3 + p_2 p_3 - 2 p_1 p_2 p_3$.

Proof: The stated probability is exactly the chance that at least two of the three variables are 1. ∎

Lemma 3.1 implies that $A_f(\frac{1}{2}) = \frac{1}{2}$ for any read-once majority formula f. Thus, $\frac{1}{2}$ is a fixed point of A_f.

Our approach depends on the fact that the first derivative of A_f is large at $\frac{1}{2}$, meaning that a slight perturbation of $D^{(1/2)}$ (i.e., the uniform distribution) tends to perturb the statistical - behavior of the formula sufficiently to allow exact identification. See Figure 1 for a graph showing the amplification function for balanced read-once majority formulas of various depths.

We perturb $D^{(1/2)}$ by hard-wiring a small number of variables to be 1; such perturbations can always be efficiently sampled by simply waiting for the desired variables to be simultaneously set to 1 in a random example from $D^{(1/2)}$.

We begin by considering the effect on the function's amplification of altering the probability with which one of the variables x_j is set to 1. This will be important in the analysis that follows.

Lemma 3.2 *Let f be a read-once majority formula, and let t be the level of an unnegated variable x_j. Then for $q \in \{0,1\}$, $A_{f|x_j \leftarrow q}(\frac{1}{2}) = \frac{1}{2} + \left(\frac{1}{2}\right)^t \left(q - \frac{1}{2}\right)$.*

Proof: By induction on t. When $t = 0$, the formula consists just of the variable x_j, and the lemma holds. For the inductive step, let f_1, f_2 and f_3 be the functions computed by the three subformulas obtained by deleting the output gate of f; thus, f is just the majority of f_1, f_2 and f_3. Note that x_j occurs in exactly one of these three subformulas—assume it occurs in the first. Since x_j occurs at level $t - 1$ of this subformula, by the inductive hypothesis, $A_{f_1|x_j \leftarrow q}(\frac{1}{2}) = \frac{1}{2} + \left(\frac{1}{2}\right)^{t-1} \left(q - \frac{1}{2}\right)$, and since x_j does not occur in the other subformulas, $A_{f_i|x_j \leftarrow q}(\frac{1}{2}) = \frac{1}{2}$ for $i = 2,3$. From Lemma 3.1, it follows that $A_{f|x_j \leftarrow q}(\frac{1}{2})$ has the stated value, completing the induction. ∎

It can now be seen how we use the amplification function to determine the relevant variables of f: if x_j is relevant, then the statistical behavior of the output of f changes significantly when x_j is hard-wired to 1. Similarly, the sign and the level of each variable can be readily determined in this manner.

Theorem 3.3 *Let f be a read-once majority formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_j \leftarrow 1}(\frac{1}{2})$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \tau \leq \left(\frac{1}{2}\right)^{h+2}$. Then*

- *x_j is relevant if and only if $|\hat{\alpha} - \frac{1}{2}| > \tau$;*

- *if x_j is relevant, then it occurs negated if and only if $\hat{\alpha} < \frac{1}{2}$;*

- *x_j occurs at level t if and only if $\left| |\hat{\alpha} - \frac{1}{2}| - \left(\frac{1}{2}\right)^{t+1} \right| < \tau$.*

Proof: This proof follows from straightforward calculations using Lemma 3.2. ∎

Thus, if one estimates the value of the amplification function from a sample whose size is polynomial in 2^h, then with high probability one can determine which variables are relevant, as well as the sign and level of every relevant variable. Specifically, we can apply Chernoff bounds (Lemma 2-3.6) to derive a sample size sufficient to ensure that all the above information is properly computed with high probability. We therefore assume henceforth that the level of every variable has been determined, and that (without loss of generality) all variables are relevant and unnegated.

More problematic is determining exactly how the variables are combined in f. A natural approach is to try hard-wiring *pairs* of variables to 1, and to again estimate the amplification of the induced function in the hopes that some structural information will be revealed. The following lemma, which is useful at a later point, shows that this approach fails.

Lemma 3.4 *Let f be a read-once majority formula, and let x_i and x_j be distinct, unnegated variables which occur at levels t_1 and t_2, respectively. Then*

$$A_{f|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1+1} + \left(\tfrac{1}{2}\right)^{t_2+1},$$

regardless of the depth d of $\lambda = \Gamma(x_i, x_j)$.

Proof: By induction on d. Let f_1, f_2 and f_3 be the three subformulas of f which are inputs to the output gate so that $f = \text{MAJ}(f_1, f_2, f_3)$.

If $d = 0$, then λ is the output gate, and x_i and x_j occur in two of the subformulas (say, f_1 and f_2, respectively). From Lemma 3.2, it follows that

$$A_{f_k|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_k}$$

for $k = 1, 2$, and, since neither x_i nor x_j is relevant to f_3, $A_{f_3|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2}$. The stated value for $A_{f|x_i,x_j \leftarrow 1}(\tfrac{1}{2})$ follows then from Lemma 3.1.

If $d > 0$, then λ is a gate occurring in one of the subformulas (say f_1) at level $d - 1$ of the subformula. By inductive hypothesis,

$$A_{f_1|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + \left(\tfrac{1}{2}\right)^{t_1} + \left(\tfrac{1}{2}\right)^{t_2}.$$

Also, $A_{f_k|x_i,x_j \leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2}$ for $k = 2, 3$. The stated value for $A_{f|x_i,x_j \leftarrow 1}(\tfrac{1}{2})$ again follows from Lemma 3.1.   ∎

Thus, if two relevant variables are hard-wired to 1, no information is obtained by knowing the value of the amplification function. That is, the amplification function is independent of the level at which the two variables meet.

Therefore, we instead consider what happens when three relevant variables of the same level are fixed to 1. In fact, it turns out to be sufficient to do so for triples of variables all of which occur at the bottom level of the formula. We show that by doing so one can determine the full structure of the formula.

For each triple x_i, x_j and x_k all occurring at level t, there are essentially two cases to consider; either

1. the triple x_i, x_j, x_k does not meet in the formula; or

2. the three variables x_i, x_j and x_k meet at the gate $\Gamma(x_i, x_j, x_k)$. We divide this case into two sub-cases:

    (a) x_i, x_j and x_k are inputs to the same gate so that $\Gamma(x_i, x_j, x_k)$ occurs at level $t - 1$; or

(b) $\Gamma(x_i, x_j, x_k)$ occurs at some level $d < t - 1$.

We are interested in separating Case 2a from the other cases by estimating the amplification of the function when all three variables are hard-wired to 1. This is sufficient to reconstruct the structure of the formula: if we can find three variables that are inputs to some gate λ (and there always must exist such a triple), then we can essentially replace the subformula consisting of the three variables and the gate λ by a new meta-variable whose value can easily be determined from the values of the original three variables. Furthermore, since $\frac{1}{2}$ is a fixed point for all read-once majority formulas the meta-variables' statistics will be the same as those of the original variables. Thus, the total number of variables is reduced by two, and the rest of the formula's structure can be determined recursively.

The following two lemmas analyze the amplification of the function when three variables are hard-wired to 1 in both of the above cases. We begin with Case 2:

Lemma 3.5 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three distinct, unnegated inputs which occur at levels t_1, t_2 and t_3, respectively, and which meet at gate $\lambda = \Gamma(x_i, x_j, x_k)$. Let d be the level of λ. Then*

$$A_{f|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + (\tfrac{1}{2})^{t_1+1} + (\tfrac{1}{2})^{t_2+1} + (\tfrac{1}{2})^{t_3+1} - (\tfrac{1}{2})^{t_1+t_2+t_3-2d-1}.$$

Proof: By induction on d. As in the preceding lemmas, suppose that $f = \text{MAJ}(f_1, f_2, f_3)$. If $d = 0$, then λ is the output gate of f, and, without loss of generality, x_i, x_j and x_k occur one each in f_1, f_2 and f_3, respectively. From Lemma 3.2, $A_{f_r|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + (\tfrac{1}{2})^{t_r}$, for $r = 1, 2, 3$. The stated value for $A_{f|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2})$ follows from Lemma 3.1.

If $d > 0$, then one of the subformulas (say, f_1) contains λ at depth $d - 1$. By inductive hypothesis,

$$A_{f_1|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + (\tfrac{1}{2})^{t_1} + (\tfrac{1}{2})^{t_2} + (\tfrac{1}{2})^{t_3} - (\tfrac{1}{2})^{t_1+t_2+t_3-2d-2},$$

and of course, $A_{f_r|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2}$ for $r = 2, 3$. The proof is completed by again applying Lemma 3.1. ∎

So, unlike the situation in which only two variables are hard-wired to 1, here the value of the amplification function depends on the level of the formula at which the three variables meet. However, it may be the case that x_i, x_j, and x_k do not meet at all (i.e., we may be in Case 1). The next lemma considers this case.

Lemma 3.6 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three distinct, unnegated inputs that occur at levels t_1, t_2 and t_3, respectively, and for which $\lambda' = \Gamma(x_i, x_j) \neq \Gamma(x_i, x_j, x_k) = \lambda$. Then*

$$A_{f|x_i,x_j,x_k\leftarrow 1}(\tfrac{1}{2}) = \tfrac{1}{2} + (\tfrac{1}{2})^{t_1+1} + (\tfrac{1}{2})^{t_2+1} + (\tfrac{1}{2})^{t_3+1},$$

regardless of the levels d and d' of gates λ and λ'.

Proof: By induction on d. As before, assume that $f = \text{MAJ}(f_1, f_2, f_3)$. If $d = 0$, then λ is the output gate, λ' occurs (say) in f_1, and x_k in f_2. From Lemma 3.4, $A_{f_1|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2}) = \frac{1}{2} + (\frac{1}{2})^{t_1} + (\frac{1}{2})^{t_2}$, and from Lemma 3.2, $A_{f_2|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2}) = \frac{1}{2} + (\frac{1}{2})^{t_3}$. Also, $A_{f_3|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2}) = \frac{1}{2}$. Lemma 3.1 then implies the stated value for $A_{f|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2})$.

If $d > 0$, then λ occurs, say, in f_1. By inductive hypothesis, $A_{f_1|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2}) = \frac{1}{2} + (\frac{1}{2})^{t_1} + (\frac{1}{2})^{t_2} + (\frac{1}{2})^{t_3}$, and clearly $A_{f_r|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2}) = \frac{1}{2}$ for $r = 2, 3$. An application of Lemma 3.1 completes the induction. ∎

Combining these lemmas, we can show that Case 2a can be separated from the other cases by estimating the function's amplification with triples of variables hard-wired to 1.

Theorem 3.7 *Let f be a read-once majority formula. Let x_i, x_j and x_k be three distinct, unnegated, level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_i,x_j,x_k\leftarrow 1}(\frac{1}{2})$ for which $|\hat{\alpha} - \alpha| < \tau \leq 3\left(\frac{1}{2}\right)^{t+4}$. Then x_i, x_j and x_k are inputs to the same gate of f if and only if $\hat{\alpha} < \frac{1}{2} + \left(\frac{1}{2}\right)^t + \tau$.*

Proof: If Case 2a applies, then Lemma 3.5 implies that $\alpha = \frac{1}{2} + \left(\frac{1}{2}\right)^t$. Otherwise, if either Case 1 or 2b applies, then Lemmas 3.5 and 3.6 imply that $\alpha \geq \frac{1}{2} + \left(\frac{1}{2}\right)^t + 3\left(\frac{1}{2}\right)^{t+3}$. The theorem follows immediately. ∎

We are now ready to state the main result of this section:

Theorem 3.8 *There exists an algorithm with the following properties: Given h, n, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0,1\}^n$ and labeled by any read-once majority formula f of depth at most h on n variables, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O(4^h \cdot \log(n/\delta))$, and its time complexity is $O(4^h \cdot (r^3 + n) \cdot \log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Proof: First, for each variable x_i, estimate the function's amplification with x_i hard-wired to 1. (We will ensure that, with high probability, this estimate is within $\left(\frac{1}{2}\right)^{h+2}$ of the true amplification.) It follows from Theorem 3.3 that after this phase of the algorithm, with high probability we know which variables are relevant, and the sign and depth of each relevant variable. (So, we assume from now on that the formula is monotone.)

In the second phase of the algorithm, we build the formula level by level from bottom to top. To build the bottom level, for all triples of variables x_i, x_j, x_k that enter the bottom level, we estimate the amplification with x_i, x_j, and x_k hard-wired to 1. (We will ensure that, with high probability, this estimate is within $3\left(\frac{1}{2}\right)^{h+4}$ of the true amplification.) It follows from Theorem 3.7 that we can determine which variables enter the same bottom-level gates.

```
LearnMajorityFormula(n, h, δ)
    E ← Θ(4^h log(n/δ)) labeled examples from D^(1/2)
    X ← ∅
    for 1 ≤ i ≤ n
        E' ← examples from E for which x_i = 1
        α̂ ← fraction of E' that are positive
        if |α̂ - ½| > (½)^(h+2) then
            if α̂ > ½ then X ← X ∪ {x_i}
                      else  X ← X ∪ {x̄_i}
            t(x_i) ← compute-level(α̂, h)
    BuildFormula(h, X, E)



BuildFormula(t, X, E)
    if t = 0 then target formula is only variable in X
    else
        for all x_i, x_j, x_k ∈ X for which t(x_i) = t(x_j) = t(x_k) = t
            E' ← examples from E for which x_i = x_j = x_k = 1
            α̂ ← fraction of E' that are positive
            if α̂ < ½ + (½)^t + 3(½)^(t+4) then
                let y ≡ MAJ(x_i, x_j, x_k) be a new variable
                t(y) ← t - 1
                X ← (X ∪ {y}) - {x_i, x_j, x_k}
        BuildFormula(t - 1, X, E)
```

Figure 2: Algorithm for exactly identifying read-once majority formulas of depth h. Procedure compute-level($α̂, h$) computes the level associated with $α̂$ as given by Theorem 3.3.

We want to recurse to compute the other levels; however, we cannot hard-wire too many variables without the filter requiring too many examples. The key observation is that on examples drawn from the uniform distribution, the output of any subformula is 1 with probability $\frac{1}{2}$. Thus, the inputs into any level are in fact distributed according to a uniform distribution. Since we compute the formula from bottom to top, the filter can just compute the value for the *known* levels to determine the inputs to the level currently being learned. Our algorithm is described in Figure 2.

Given that the estimates for the amplification function have the needed accuracy, the proof of correctness follows from Theorems 3.3 and 3.7. Specifically, for each variable x_i, we need a good estimate $α̂$ of $α = A_{f|x_i, -1}(\frac{1}{2})$; we require that the chosen sample be sufficiently large that $|α - α̂| < 2^{-(h+2)}$ with probability at least $1 - δ/2n$. Then every such estimate for the n variables will have the needed accuracy with probability at least $1 - δ/2$. These estimates thus satisfy the requirements of the first phase of the algorithm.

In the second phase, we require good estimates of the formula's amplification when triples of variables are hard-wired. In fact, we need such estimates not only when ordinary variables are hard-wired, but also when we hard-wire meta-variables. Note that, assuming all estimates have the needed accuracy, every (meta-)variable added to the set X in Figure 2 in fact computes some subformula g of f. Thus, for every triple of subformulas g_1, g_2 and g_3 of f, our algorithm requires an estimate $\hat{\alpha}$ of α, the amplification of f at $\frac{1}{2}$, given that the output of each subformula g_1, g_2 and g_3 is fixed to the value 1. Since a read-once majority formula on n variables has at most $3n/2$ subformulas (since it has at most $n/2$ MAJ gates), we require a sample sufficiently large that $|\alpha - \hat{\alpha}| < 3 \left(\frac{1}{2}\right)^{h+4}$ with probability at least $1 - 4\delta/(3n)^3$. The chance that all of the (at most $(3n/2)^3$) estimates have the needed accuracy is then at least $1 - \delta/2$.

Thus, a sufficiently large sample provides all of the needed estimates with probability at least $1 - \delta$. The sample size required can be derived using a standard application of Chernoff bounds (Lemma 2-3.6).   ∎

Note that our algorithm's sample complexity has only a logarithmic dependence on the number of irrelevant attributes. Also, it follows immediately from Theorem 3.8 that any read-once majority formula of depth $O(\log n)$ can be exactly identified in polynomial time.

Finally, we note that our algorithm can be modified to work without receiving a bound for the height of the formula as input; the time and sample complexity only increase by a factor of two. The idea is to guess an initial value of $h = 1$ and to increment our guess each time the algorithm fails; it can be shown that, if the formula's height is greater than our current guess, then this fact will become evident by our algorithm's inability to successfully construct a formula. (Specifically, the algorithm BuildFormula in Figure 2 will reach a point at which there remain level-t variables in X, but no three remaining level-t variables are immediate inputs to the same gate.)

3-4   Exact identification of read-once positive NAND formulas

In this section we use the properties of the amplification function to obtain a polynomial-time algorithm that with high probability exactly identifies any read-once positive NAND formula of logarithmic depth from $D^{(\psi)}$ where ψ is the constant $(\sqrt{5} - 1)/2 \approx 0.618$. Note that $\psi = 1/\phi = \phi - 1$, where ϕ is the golden ratio.

The class of read-once positive NAND formulas is equivalent to the class of read-once formulas constructed from alternating levels of OR/AND gates, starting with an OR gate at top level, and with the additional condition that each variable is negated if and only if it enters an OR gate. This observation is easily proved by repeated application of DeMorgan's law.

It is interesting to compare our result with what is known about learning this class of formulas in other models. It follows from the results of Kearns and Valiant [52] and Pitt and

Warmuth [70] that learning this class of formulas is hard in the distribution-free model (under cryptographic assumptions). Thus, there exist distributions that reveal essentially no information about the formula that is useful for prediction to a computationally-bounded algorithm. If one views the sampling of the distribution $D^{(\psi)}$ as a form of non-adaptive "random membership queries," our result can also be compared with the algorithm of Angluin, Hellerstein and Karpinski [8] which uses membership and equivalence queries that are considerably more complicated and are highly dependent on the target concept; on the other hand, their algorithm can be used to identify a broader class of formulas.

We show that this class of formulas is learnable when examples are chosen from a distribution in which each variable is 1 with probability ψ. The basic structure of the algorithm is just like that of the preceding algorithm for identifying read-once majority formulas. In the first phase of the algorithm, we determine the relevant variables and their depths by hard-wiring each variable to 0, and estimating the amplification of the induced function at ψ using random examples from $D^{(\psi)}$. In the second phase of the algorithm, we construct the formula by finding pairs of variables that are direct inputs to a bottom-level gate. Here, we show that this is possible by hard-wiring pairs of variables to 0 and estimating the function's amplification. After learning the structure of the bottom level of the formula, we again are able to construct the remaining levels recursively.

Since the techniques used in this section are so similar to those in Section 3-3, the proofs of the lemmas and theorems have been omitted. Most of the lemmas can be proved by simple induction arguments as before.

We turn now to a discussion of some of the properties of the amplification function of read-once positive NAND formulas; these lead to a proof of the correctness of our algorithm.

Lemma 4.1 *Let X_1 and X_2 be independent Bernoulli variables, each 1 with probability p_1 and p_2, respectively. Then* $\Pr[\text{NAND}(X_1, X_2) = 1] = 1 - p_1 p_2$.

It is easily verified that $1 - \psi^2 = \psi$, and thus that ψ is a fixed point of the amplification function A_f whenever f is a read-once positive NAND formula. Once again, our approach depends on the fact that slight perturbations of $D^{(\psi)}$ tend to perturb the statistical behavior of the formula sufficiently to allow exact identification.

Lemma 4.2 *Let f be a read-once positive NAND formula, and let t be the level of some variable x_j. Then* $A_{f|x_j \leftarrow q}(\psi) = \psi + (q - \psi)(-\psi)^t$.

Thus, hard-wiring an even-leveled input to 0 decreases the amplification while hard-wiring an odd-leveled input to 0 increases the amplification. To give some intuition explaining this behavior, consider the correspondence described above between read-once positive NAND formulas and leveled OR/AND formulas. An even-leveled input corresponds to an input to an AND

gate and thus hard-wiring that input to 0 clearly decreases the amplification. However, an odd-leveled input corresponds to an input that is first negated and then fed to an OR gate; thus, this case corresponds to hard-wiring the input to an OR gate to 1 which clearly increases the amplification function.

As we saw in the last section, the amplification function can be used to determine the relevant variables of f: if x_j is relevant then the statistical behavior of the output of f changes significantly when x_j is hard-wired to 0. Similarly, the level of each variable can be computed in this manner.

Theorem 4.3 *Let f be a read-once positive* NAND *formula of depth h. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_j \leftarrow 0}(\psi)$ for some variable x_j, and assume that $|\hat{\alpha} - \alpha| < \tau \leq \psi^{h+1}/2$. Then*

- x_j *is relevant if and only if $|\hat{\alpha} - \psi| > \tau$;*

- x_j *occurs at level t if and only if $|\psi + (-\psi)^{t+1} - \hat{\alpha}| < \tau$.*

We next consider the effect on the amplification function of hard-wiring two inputs. Unlike the case of majority formulas, measuring the amplification of the function when pairs of variables are hard-wired to 0 reveals a great deal of information about the structure of the formula. In particular, the value of the amplification function when two level-t variables x_i and x_j are hard-wired to 0 depends critically on the depth of $\Gamma(x_i, x_j)$.

Lemma 4.4 *Let f be a read-once positive* NAND *formula, and let x_i and x_j be two distinct variables which occur at levels t_1 and t_2, respectively, and for which $\lambda = \Gamma(x_i, x_j)$ is at level d. Then*

$$A_{f|x_i, x_j \leftarrow 0}(\psi) = \psi + (-\psi)^{t_1+1} + (-\psi)^{t_2+1} - (-\psi)^{t_1+t_2-d}.$$

Using the same ideas as in the last section, it can now be proved that, given a good estimate of the amplification function, one can determine which variables meet at bottom-level gates.

Theorem 4.5 *Let f be a read-once positive* NAND *formula. Let x_i and x_j be two level-t inputs. Let $\hat{\alpha}$ be an estimate of $\alpha = A_{f|x_i, x_j \leftarrow 0}(\psi)$ for which $|\hat{\alpha} - \alpha| < \tau \leq \psi^{t+3}/2$. Then x_i and x_j are inputs to the same level-t gate of f if and only if $|\psi + (-\psi)^{t+1} - \hat{\alpha}| < \tau$.*

We are now ready to state the main result of this section:

Theorem 4.6 *Let $\psi = 1/\phi = (\sqrt{5} - 1)/2$. Then there exists an algorithm with the following properties: Given h, n, $\delta > 0$, and access to examples drawn from the distribution $D^{(\psi)}$ on $\{0,1\}^n$ and labeled by any read-once positive* NAND *formula f of depth at most h on n variables, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O(\phi^{2h} \cdot \log(n/\delta))$, and its time complexity is $O(\phi^{2h} \cdot (r^2 + n) \cdot \log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Our algorithm is obtained by making the obvious modifications to `LearnMajorityFormula` and `BuildFormula`. The proof that this algorithm is correct follows from the preceding lemmas and theorems, and is similar to the proof of Theorem 3.8.

As before, it follows immediately that any read-once positive NAND formula of depth at most $O(\log n)$ can be exactly identified in polynomial time.

3-5   Handling random misclassification noise

Because the algorithms described in the preceding sections are statistical in nature, they are easily modified to handle a considerable amount of noise. In this section, we describe a robust version of our algorithm for learning logarithmic-depth read-once majority formulas. Although omitted, a similar (though slightly more involved) algorithm can be derived for NAND formulas.

Our algorithm is able to handle a kind of random misclassification noise which is similar, but slightly more general than that considered by Angluin and Laird [9], and Sloan [81]. Specifically, the output of the target formula is "flipped" with some fixed probability which may depend on the formula's output. Thus, if the true, computed output of the formula is 0, then the learner sees 0 with probability $1 - \eta_0$, and 1 with probability η_0, for some quantity η_0. Similarly, a true output of 1 is observed to be 0 with probability η_1 and 1 with probability $1 - \eta_1$. When $\eta_0 = \eta_1$, this noise model is equivalent to that considered by Angluin and Laird, and Sloan. Note that when $\eta_0 + \eta_1 = 1$, outputs of 0 or 1 are entirely indistinguishable in an information-theoretic sense. Moreover, we can assume without loss of generality that $\eta_0 + \eta_1 \leq 1$ by symmetry of the behavior of the formula f with its negation $\neg f$.

If we regard our algorithm's use of a fixed distribution as a form of membership query, we can also handle large rates of misclassification noise in the queries. Here the formulation of a meaningful noise model is more problematic. In particular, we wish to disallow the uninteresting technique of repeatedly querying a particular instance in order to obtain its true classification with overwhelming probability. Thus, we consider a model in which noisy labels are *persistent:* for each instance x, on the first query to x, the true output of the target concept is computed and is reversed with probability η_0 or η_1, according to whether the true output is 0 or 1 (as described above). However, on all subsequent queries to x, the label returned is the same as the label returned with the first query to x. A natural interpretation of such persistent noise is that of a teacher who is simply wrong on certain instances, and cannot be expected to change his mind with repeated sampling. This kind of persistent noise is not a problem for our algorithms because, when n is large, the algorithm is extremely unlikely to query the same instance twice.

Our algorithm assumes that $\eta_0 + \eta_1$ is bounded away from 1 so that $\eta_0 + \eta_1 \leq 1 - \rho$ for some known positive quantity ρ. The error rates themselves, η_0 and η_1, are assumed to be unknown. Our algorithm exactly identifies the target formula with high probability in time polynomial in

all of the usual parameters, and $1/\rho$.

Our robust algorithm has a similar structure to that of the algorithm described previously for the noise-free case: The algorithm begins by determining the relevance and sign of each variable. However, it is not clear at this point how the level of each variable might be ascertained in the presence of noise. Nevertheless, it turns out to be possible to find three bottom-level variables which are inputs to the same gate. As before, once such a triple has been discovered, the remainder of the formula can be identified recursively.

To start with, note that if p is the probability that a 1 is output by the target formula f under some distribution on $\{0,1\}^n$, then the probability that a 1 is observed by the learner is

$$p(1 - \eta_1) + (1 - p)\eta_0 = p(1 - \eta_0 - \eta_1) + \eta_0 \ .$$

Thus, $\tilde{A}_f(p) = A_f(p) \cdot (1 - \eta_0 - \eta_1) + \eta_0$ is the probability that a 1 is observed when each input is 1 with probability p. Under the uniform distribution, a 1 is observed with probability $\xi = \tilde{A}_f(\frac{1}{2})$. Since η_0 and η_1 are unknown, ξ is unknown as well. However, an accurate estimate $\hat{\xi}$ (say, within $\Theta(\rho/2^h)$ of ξ) can be efficiently obtained in the usual manner by sampling.

The next lemma shows that a variable x_i's relevance and sign can be determined by hard-wiring it to 1 and comparing $\hat{\xi}$ to an estimate of the value $\alpha = \tilde{A}_{f|x_i \leftarrow 1}(\frac{1}{2})$.

Lemma 5.1 *Let f be read-once majority formula of depth h. Let $\hat{\xi}$ and $\hat{\alpha}$ be estimates of $\xi = \tilde{A}_f(\frac{1}{2})$ and $\alpha = \tilde{A}_{f|x_j \leftarrow 1}(\frac{1}{2})$, for some variable x_j. Assume $|\alpha - \hat{\alpha}| < \tau$ and $|\xi - \hat{\xi}| < \tau$ for some $\tau \leq \rho/2^{h+3}$. Then*

- *x_j is relevant if and only if $|\hat{\alpha} - \hat{\xi}| > 2\tau$;*

- *if x_j is relevant, then it occurs negated if and only if $\hat{\alpha} < \hat{\xi}$.*

Proof: Note that $\alpha - \xi = (1 - \eta_0 - \eta_1)(A_{f|x_j \leftarrow 1}(\frac{1}{2}) - \frac{1}{2})$. The lemma then follows from Lemma 3.2, and by noting that $1 - \eta_0 - \eta_1 \geq \rho$. ∎

More difficult is the problem of determining the level of each variable since η_0 and η_1 are unknown. Nevertheless, it turns out to be possible to identify the formula without first determining the level of each variable. In particular, we can determine a triple of variables which are inputs to the same bottom-level gate. As described in Section 3-3, once this is done, the three variables can be replaced by a meta-variable, and the rest of the formula can be constructed recursively. Thus, to complete the algorithm, we need only describe a technique for finding such a triple.

From the comments above, we can assume without loss of generality that all variables are relevant and unnegated. The key point, proved below, is the following: $\tilde{A}_{f|x_i, x_j, x_k \leftarrow 1}(\frac{1}{2})$ is minimized over triples x_i, x_j and x_k whenever the three variables are inputs to the same

bottom-level gate. Thus, such a triple can be found by estimating $\bar{A}_{f|x_i,x_j,x_k=1}(\frac{1}{2})$ for each triple and choosing the one with the smallest estimated value.

Lemma 5.2 *Let f be a monotone, read-once majority formula of depth h. For all triples of distinct indices i, j and k, let $\hat{\alpha}_{ijk}$ be an estimate of $\alpha_{ijk} = \bar{A}_{f|x_i,x_j,x_k=1}(\frac{1}{2})$, and assume that $|\alpha_{ijk} - \hat{\alpha}_{ijk}| < \tau \le 3\rho/2^{h+4}$. Suppose that $\hat{\alpha}_{qrs} = \min\{\hat{\alpha}_{ijk} : i,j,k \text{ distinct }\}$. Then x_q, x_r and x_s are bottom-level variables that are inputs to the same gate.*

Proof: From Lemma 3.5, if x_i, x_j and x_k are bottom-level variables that are inputs to the same gate, then

$$\alpha_{ijk} = (1 - \eta_0 - \eta_1)(\tfrac{1}{2} + (\tfrac{1}{2})^h) + \eta_0.$$

Otherwise, Lemmas 3.5 and 3.6 imply that

$$\alpha_{ijk} \ge (1 - \eta_0 - \eta_1)(\tfrac{1}{2} + (\tfrac{1}{2})^h) + \eta_0 + 3\rho/2^{h+3}.$$

Since each $\hat{\alpha}_{ijk}$ is accurate to within $3\rho/2^{h+4}$, it follows that $\hat{\alpha}_{qrs}$ can be minimal only if x_q, x_r and x_s are bottom-level inputs to the same gate. ∎

Thus, Lemma 5.2 gives a technique for finding bottom-level inputs to the same gate, and, as previously mentioned, the remainder of the formula can be constructed recursively as in Section 3-3. We thus obtain the main result of this section:

Theorem 5.3 *There exists an algorithm with the following properties: Given h, n, $\rho > 0$, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0,1\}^n$, labeled by a read-once majority formula f of depth at most h on n variables, and misclassified with probabilities η_0 and η_1 (as described above) for $\eta_0 + \eta_1 \le 1 - \rho$, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O((4^h/\rho^2) \cdot \log(n/\delta))$, and its time complexity is $O((4^h/\rho^2) \cdot (n + r^3) \cdot \log(n/\delta))$, where r is the number of relevant variables appearing in the target formula.*

Finally, we comment that our algorithms can be extended to handle a fair amount of *malicious noise*. In this model, an adversary is allowed to corrupt each example in any manner he chooses (both the labels and the variable settings) with probability η. We can show that the algorithm described in Sections 3-3 for majority formulas can handle malicious error rates as large as $\Theta(2^{-h})$ where h is the height of the target formula. Thus, for logarithmic-depth formulas, we can handle malicious error rates up to an inverse polynomial in the number of relevant variables. Similar results also hold for NAND formulas.

The extension of the algorithm to handle malicious noise is quite simple. The algorithm of Section 3-3 depends only on accurate estimates of probabilities α that the formula outputs

1 under various distributions. Note that malicious noise can change such probabilities by at most an additive factor of η. That is, if the formula, under some distribution, outputs 1 with probability α, then the chance that a positive example is observed (in the presence of malicious noise) is at least $\alpha - \eta$ and at most $\alpha + \eta$.

Thus, using Chernoff bounds (Lemma 2-3.6), an estimate of α that is accurate to within $\eta + \tau$ can be obtained from a sample of size polynomial in $1/\tau$ (with high probability). Since Theorems 3.3 and 3.7 show that the required estimates need only be accurate to within $3/2^{h+4}$, it follows that a malicious error rate of, say, half this amount can be tolerated without increasing the algorithm's complexity by more than constant factors.

3-6   Learning unbounded-depth formulas

In this final section on amplification-function techniques, we describe extensions of our algorithms to learn formulas of unbounded depth in Valiant's PAC model with respect to specific distributions. As in the last section, we focus only on majority formulas, omitting the similar application of these techniques to NAND formulas.

For formulas of unbounded depth, exact identification from the uniform distribution in polynomial time is too much to ask: For purely information-theoretic reasons, at least $\Omega(2^h)$ examples must be drawn from the uniform distribution to exactly identify a majority formula of depth h. This can be proved by showing (say, by induction on h) that if x_i occurs at level h of formula f, then 2^{-h} is the probability that an instance is chosen for which the output of f depends on x_i (i.e., for which f's output changes if x_i is flipped). Thus, $\Omega(2^h)$ random examples are needed simply to determine, for example, whether x_i occurs negated or unnegated.

Therefore, to handle arbitrarily deep formulas, we must relax our requirement of exact identification. Instead, we adopt Valiant's criterion of obtaining a good *approximation* of the target concept (with high probability). As before, our algorithms do not work for all distributions, just the fixed-point distribution. We describe an algorithm that, given $\epsilon, \delta > 0$ and access to random examples of the target majority formula drawn from the uniform distribution, outputs with probability $1 - \delta$ an ϵ-*good* hypothesis, that is, one that agrees with the target formula on a randomly chosen instance from the uniform distribution with probability at least $1 - \epsilon$. Furthermore, the running time is polynomial in $1/\delta$, $1/\epsilon$ and the number of variables n.

We begin by briefly discussing the main ideas of the algorithm. First, as noted above, variables that occur deep in the formula are unimportant in the sense that their values are unlikely to influence the formula's output on a randomly chosen instance. Intuitively, we would like to take advantage of this fact by somehow treating such variables as irrelevant. However, they cannot be simply deleted from the formula without leaving "holes" that must in some way be handled.

We therefore introduce the notion of a *partially visible function*. This is a function on a set of *visible* variables whose values can be observed by the learner, and a set of *hidden* variables which are not observable. With respect to a distribution on the set of assignments to the hidden variables, we say that two partially visible Boolean functions are *equivalent* if, for all assignments to the visible variables, the probabilities are the same that each function evaluates to 1 (where the probabilities are taken over random assignments to the hidden variables). In other words, the behaviors of the two functions are indistinguishable with respect to the visible variables.

Thus, we handle all deep variables by regarding them as hidden variables, and the target formula as one that is partially visible. In particular, *insignificant* variables—those that occur below level $h = \lceil \lg(n/2\epsilon) \rceil$—are considered hidden and their actual values ignored. We call the partially visible formula obtained from the target formula f in this manner the *truncated target*.

Our algorithm works by exactly identifying the truncated target, that is, by constructing a partially visible formula f' that is equivalent to it (in the sense described above, with respect to the uniform distribution). It can be shown that f and f' agree on a randomly chosen instance with probability at least $1 - \epsilon$, and therefore f' is an ϵ-good hypothesis satisfying the PAC criterion.

It remains then only to show how f' can be constructed. First, observe that by Lemma 3.2 all significant variables occurring in f can be detected (and their signs and levels determined) in polynomial time. Moreover, by arguments similar to those given in Section 3-3, it can be shown that if some triple of significant variables meet at a gate, then the level of that gate can be detected from the amplification function by hard-wiring the three variables to 1. We call this information (the level and sign of each significant variable, and the level at which each triple of significant variables meet, if at all) the formula's *schedule*. It turns out that the schedule alone is sufficient to fully re-construct the partially visible formula f', as is shown below.

These then are the main ideas of the algorithm. What follows is a more detailed exposition.

A *partially visible function* $f(x : y)$ is a Boolean function f on a set of *visible variables* $x = x_1 \cdots x_r$, and a set of *hidden variables* $y = y_1 \cdots y_s$. Two partially visible functions $f(x : y)$ and $g(x : z)$ on the same set of visible variables are *equivalent* with respect to distributions D and E on the domains of y and z if, for all x, $\Pr[f(x : Y) = 1] = \Pr[g(x : Z) = 1]$, where Y and Z are random variables representing a random assignment to y and z according to D and E. In the discussion that follows, we will only be interested in uniform distributions.

As described above, our algorithm regards variables that occur deep in the target formula as hidden variables. The next two lemmas show that two partially visible read-once majority formulas that are identical except for some deep hidden variables are very likely to produce the

same output on randomly chosen inputs.

Lemma 6.1 *Let f be a read-once majority formula on n variables. Let t be the level of x_n in f. Let $X_1, \ldots, X_{n-1}$, Y and Z be independent Bernoulli variables, each 1 with probability $1/2$. Then $\Pr[f(X_1, \ldots, X_{n-1}; Y) \neq f(X_1, \ldots, X_{n-1}; Z)] = 2^{-t-1}$.*

Proof: By induction on t. If $t = 0$, then f is the function x_n and since $\Pr[Y \neq Z] = 1/2$, the lemma holds. If $t > 0$, then let $f = \text{MAJ}(f_1, f_2, f_3)$, and suppose that f_1 is the subformula in which x_n occurs. Since x_n does not also occur in f_2 or f_3, we will regard these as functions only on the remaining $n - 1$ variables. It is not hard to see then that $f(X_1, \ldots, X_{n-1}; Y) \neq f(X_1, \ldots, X_{n-1}; Z)$ if and only if $f_2(X_1, \ldots, X_{n-1}) \neq f_3(X_1, \ldots, X_{n-1})$ and $f_1(X_1, \ldots, X_{n-1}; Y) \neq f_1(X_1, \ldots, X_{n-1}; Z)$. Since f_2 and f_3 each output 1 independently with probability $1/2$, we have

$$\Pr[f_2(X_1, \ldots, X_{n-1}) \neq f_3(X_1, \ldots, X_{n-1})] = 1/2.$$

Also, by inductive hypothesis,

$$\Pr[f_1(X_1, \ldots, X_{n-1}; Y) \neq f_1(X_1, \ldots, X_{n-1}; Z)] = 2^{-t}.$$

The lemma then follows by independence.   ∎

Lemma 6.2 *Let f be a read-once majority formula on n variables. Let t_i be the level of variable x_i in f. Let $X_1, \ldots, X_n; X_1', \ldots, X_r'$, $r \leq n$, be independent Bernoulli variables, each 1 with probability $1/2$. Then $\Pr[f(X_1, \ldots, X_n) \neq f(X_1', \ldots, X_r', X_{r+1}, \ldots, X_n)] \leq \sum_{i=1}^r 2^{-t_i-1}$.*

Proof: By induction on r. If $r = 0$, then the lemma holds trivially. For $r > 0$, we have

$$\begin{aligned} &\Pr[f(X_1, \ldots, X_n) \neq f(X_1', \ldots, X_r', X_{r+1}, \ldots, X_n)] \\ &\leq \quad \Pr[f(X_1, \ldots, X_n) \neq f(X_1', \ldots, X_{r-1}', X_r, \ldots, X_n)] \\ &\quad + \Pr[f(X_1', \ldots, X_{r-1}', X_r, \ldots, X_n) \neq f(X_1', \ldots, X_r', X_{r+1}, \ldots, X_n)] \\ &\leq \quad \sum_{i=1}^{r-1} 2^{-t_i-1} + 2^{-t_r-1} \end{aligned}$$

where the last inequality follows from our inductive hypothesis and the preceding lemma.   ∎

As proved below, Lemma 6.2 implies that any partially visible formula is an ϵ-good hypothesis if it is equivalent to the truncated target, the partially visible formula obtained from the target formula by regarding all variables at or below level $h = \lceil \lg(n/2\epsilon) \rceil$ as hidden variables. Given an assignment to the visible variables, such a hypothesis is evaluated in the obvious

manner by choosing a random assignment to the hidden variables and computing the output of the formula on the combined assignments to the hidden and visible variables. (Thus, the hypothesis is likely to be randomized.)

Lemma 6.3 *Let $\epsilon > 0$, and let f be a read-once majority formula on n variables. Let $x = x_1 \cdots x_r$ be the variables occurring above level $h = \lceil \lg(n/2\epsilon) \rceil$, and let $y = y_1 \cdots y_{n-r}$ be the remaining variables. Let $g(x : z)$ be any partially visible formula equivalent to the partially visible formula $f(x : y)$. Then $Pr[f(X : Y) \neq g(X : Z)] \leq \epsilon$, where X, Y and Z are random variables representing the uniformly random choice of assignments to x, y and z. That is, $g(x : z)$ is an ϵ-good hypothesis for f.*

Proof: Let Y' be a random variable representing a random assignment to y, chosen independently of Y. Since $f(x : y)$ is equivalent to $g(x : z)$, we have

$$\mathbf{Pr}[f(X : Y) \neq g(X : Z)] = \mathbf{Pr}[f(X : Y) \neq f(X : Y')].$$

By Lemma 6.2, the right hand side of this equation is bounded by ϵ, since each of the $n - r \leq n$ variables y_i occurs at or below level h in f. ∎

For $\epsilon > 0$ and target formula f, we will henceforth say that variables occurring above level $h = \lceil \lg(n/2\epsilon) \rceil$ are *significant*. Note that Theorem 3.3 implies that the significance, sign and level of any variable x_j can be determined by hard-wiring that variable to 1, as usual. More specifically, if $\hat{\alpha}$ is an estimate of $\alpha = A_{f|x_j \leftarrow 1}(\frac{1}{2})$ for which $|\hat{\alpha} - \alpha| < \tau \leq \left(\frac{1}{2}\right)^{h+2}$ then x_j is significant if and only if $|\hat{\alpha} - \frac{1}{2}| > \left(\frac{1}{2}\right)^{h+1} + \tau$, and, if it is significant, then its sign and level can be determined as in Theorem 3.3.

Similar to Theorem 3.7, we can show that, for any triple of significant variables, we can determine the level of the gate at which the triple meets, if at all. More precisely, if x_i, x_j and x_k are three unnegated variables occurring at levels t_1, t_2 and t_3, and if $\hat{\alpha}$ is an estimate of $\alpha = A_{f|x_i,x_j,x_k \leftarrow 1}(\frac{1}{2})$ for which $|\hat{\alpha} - \alpha| < \tau \leq 2^{-3h}$, then it follows from Lemmas 3.5 and 3.6 that x_i, x_j and x_k meet at a level-d gate if and only if

$$\left| \frac{1}{2} + \left(\frac{1}{2}\right)^{t_1+1} + \left(\frac{1}{2}\right)^{t_2+1} + \left(\frac{1}{2}\right)^{t_3+1} - \left(\frac{1}{2}\right)^{t_1+t_2+t_3-2d-1} - \hat{\alpha} \right| < \tau.$$

As mentioned above, we call the sum total of this information—the significance of each variable, the level and sign of each significant variable, and the level of the gate at which each triple of significant variables meet, if at all—the formula's *schedule*. It remains then only to show how an ϵ-good hypothesis can be constructed from the schedule. Specifically, we show how to construct a partially visible formula that is equivalent to the truncated target $f(x : y)$. (Here, x is the vector of visible (i.e., significant) variables, and y is the vector of hidden (insignificant) variables.)

Suppose first that no three visible variables meet in f. Such a formula is said to be *unstructured*. Although strictly speaking f cannot be unstructured (by our choice of h), this special case turns out nevertheless to be important in handling the more general case since *subformulas* of f may be unstructured.

Lemma 6.4 below shows that an unstructured formula $f(x : y)$ is equivalent to any other unstructured partially visible formula whenever each visible variable occurs at the same level with the same sign in both formulas. Thus, unstructured formulas are not changed when visible variables are moved around within the same level. This fact makes the identification of unstructured formulas from their schedules quite easy.

For any partially visible read-once majority formula $f(x : y)$, let $p_f(x) = \Pr[f(x : Y) = 1]$ where Y represents a random assignment to y.

Lemma 6.4 *Let $f(x : y)$ and $g(x : z)$ be unstructured read-once majority formulas on s visible variables. Suppose that each visible variable x_j is relevant and occurs at the same level t_j with the same sign in both formulas. Then the two partially visible formulas are equivalent.*

Proof: It suffices to prove the lemma when no visible variable is negated since negated variables can simply be replaced by unnegated meta-variables.

To prove the lemma, we show that

$$p_f(x) = \tfrac{1}{2} + \sum_{i=1}^{s} 2^{-t_i}(x_i - \tfrac{1}{2}). \tag{6.1}$$

Since this statement applies to any unstructured formula, it follows immediately that $p_f(x) = p_g(x)$ and the two partially visible formulas are equivalent.

We prove Equation (6.1) by induction on the height h of f. If $h = 0$, then f consists of a single visible or hidden variable. If f is the formula x_j, where x_j is some visible variable, then $p_f(x) = x_j$, satisfying (6.1). If f is the formula y_j, where y_j is a hidden variable, then $p_f(x) = \tfrac{1}{2}$, also satisfying (6.1).

If $h > 0$, then let $f = \text{MAJ}(f_1, f_2, f_3)$ where f_1, f_2 and f_3 are partially visible subformulas. Since f is unstructured, one of these (say f_3) contains no visible variables, and thus $p_{f_3}(x) = \tfrac{1}{2}$. Suppose without loss of generality that $x_1, \ldots, x_r$ are the visible variables relevant to f_1. Then, by inductive hypothesis,

$$p_{f_1}(x) = \tfrac{1}{2} + \sum_{i=1}^{r} 2^{-t_i+1}(x_i - \tfrac{1}{2})$$

and

$$p_{f_2}(x) = \tfrac{1}{2} + \sum_{i=r+1}^{s} 2^{-t_i+1}(x_i - \tfrac{1}{2}).$$

Applying Lemma 3.1, it is easily verified that Equation (6.1) is satisfied, completing the induction. ∎

Thus, if $f(x : y)$ is unstructured, then an equivalent unstructured formula can be constructed from f's schedule. For instance, here is an efficient algorithm: Let t be the depth of the deepest visible variable in f. Break the set of all level-t variables into pairs. Replace each such pair x_i, x_j by a level-$(t-1)$ meta-variable $w \equiv \text{MAJ}(x_i, x_j, y)$, where y is a new hidden variable. If an odd level-t variable x_i remains, replace it with a level-$(t-1)$ meta-variable $w \equiv \text{MAJ}(x_i, y, y')$, where y and y' are new hidden variables. Repeat for levels $t-1, t-2, \ldots, 1$. It is not hard to show that this algorithm results in a formula that is unstructured, and that is consistent with f's schedule (and so is equivalent).

With these tools in hand for dealing with unstructured formulas, we are now ready to describe an algorithm for handling the general case, i.e., for reconstructing any (not necessarily unstructured) formula from its schedule.

Let $f(x : y)$ be the truncated target. If f is unstructured, then the previous algorithm applies. Otherwise, we can find from the schedule three visible variables x_i, x_j and x_k which meet at some maximum-depth gate λ of f; that is, they meet at a level-d gate, and no triple of visible variables meet at any gate of depth exceeding d. Then the subformula g subsumed by λ computes the majority of three subformulas g_1, g_2 and g_3, each containing one of x_i, x_j and x_k (say, in that order). Let x_ℓ be some other visible variable. Then it is easily verified that x_ℓ is relevant to g_1 if and only if x_ℓ, x_j and x_k meet at a level-d gate (namely, λ). Thus, all of the visible variables relevant to g_1 (and likewise for g_2 and g_3) can be determined from the schedule. Moreover, note that each of these subformulas is unstructured since λ is of maximum depth. Thus, each subformula can be identified using the previous algorithm for unstructured formulas, and therefore, the entire subformula subsumed by (and including) λ can be identified.

The rest of the formula can be identified recursively: we replace subformula g by a new meta-variable w, and update the schedule appropriately.

This completes the algorithm. The sample complexity can be derived, as usual, using Chernoff bounds (Lemma 2-3.6), and the time analysis is straightforward. We thus have:

Theorem 6.5 *There exists an algorithm with the following properties: Given n, $\delta > 0$, and access to examples drawn from the uniform distribution on $\{0,1\}^n$ and labeled by any read-once majority formula f on n variables, the algorithm exactly identifies f with probability at least $1 - \delta$. The algorithm's sample complexity is $O((n/\epsilon)^6 \cdot \log(n/\delta))$, and its time complexity is $O((n^9/\epsilon^6) \cdot \log(n/\delta))$.*

3-7   Learning probabilistic read-once formulas

In this section, we extend the techniques of the preceding sections to a broad class of *probabilistic concepts*, which includes the class of all read-once formulas over the usual basis $\{\text{AND}, \text{OR}, \text{NOT}\}$. We show this class is PAC learnable against all product distributions (i.e., all distributions in which the assignment to each variable is independent of the settings of the other variables).

As described in detail in Chapter 4, a *probabilistic concept (p-concept)* is a function $c : X \rightarrow [0,1]$ where X is the domain. (In this chapter, X is always $\{0,1\}^n$.) The interpretation here is that $c(x)$ is the probability that an instance $x \in X$ is labeled 1, and $1 - c(x)$ is the probability it is labeled 0. Thus, we assume an oracle EX which first chooses $x \in X$ randomly according to some target distribution D, and then randomly labels x according to c as just described.

In this section, we will be interested in the problem of *learning with a model of probability*. Here, the goal is to infer a good approximation of the function c itself. Specifically, given positive ϵ and δ, we ask that, with probability at least $1 - \delta$, the learning algorithm find an *ϵ-good model of probability* for f, i.e., a real-valued hypothesis h such that

$$\mathbf{E}_{x \in D}\left[|h(x) - c(x)|\right] \leq \epsilon. \tag{7.1}$$

Furthermore, the learning algorithm's running time must be polynomial in $1/\epsilon$, $1/\delta$ and n. (This definition differs slightly, but is equivalent to, the definition of learning with a *model of probability* given in Chapter 4. See Section 4-2.)

We describe in this section an algorithm for learning with a model of probability any p-concept in a particular class of p-concepts against any *product distribution* on the domain $\{0,1\}^n$, i.e., any distribution in which the setting of each bit x_i is chosen independently of the settings of the other bits.

The p-concept class of interest is the class of real-valued read-once formulas over the basis $\{\text{MUL}, \text{LIN}_{zw}\}$ where MUL denotes ordinary multiplication of two real numbers, and LIN_{zw} is the unary operator

$$\text{LIN}_{zw}(y) = z + wy.$$

Here, z and w may be any real numbers for which z and $z + w$ are both in the range $[0,1]$. We call formulas over this basis *real formulas*.

For instance,

$$\text{LIN}_{0,.25}(\text{MUL}(\text{LIN}_{.5,.5}(x_1), \text{LIN}_{1,-1}(x_2))) = .25 \cdot (.5 + .5x_1) \cdot (1 - x_2) \tag{7.2}$$

is a read-once real formula.

An easy induction argument shows that real formulas have range $[0,1]$, and so are p-concepts.

Also, note that for Boolean-valued inputs, MUL is equivalent to AND, and $\text{LIN}_{1,-1}$ is equivalent to NOT. Thus, the class of read-once real formulas includes the class of read-once Boolean formulas over the basis $\{\text{AND}, \text{NOT}\}$, or, equivalently, $\{\text{AND}, \text{OR}, \text{NOT}\}$.

Further, our criterion (7.1) for good learning of real formulas implies the usual PAC learnability of Boolean formulas: if c is a deterministic concept (i.e., a p-concept with range in $\{0, 1\}$), and h is a real-valued hypothesis satisfying (7.1), then $h' = \text{round}(h) = \lfloor h + 1/2 \rfloor$ is a 2ϵ-good hypothesis for c since

$$\mathbf{Pr}_{x \in D}\left[h'(x) \neq c(x)\right] \leq \mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| \geq 1/2\right]$$

and, by Markov's inequality,

$$\epsilon \geq \mathbf{E}_{x \in D}\left[|h(x) - c(x)|\right] \geq \tfrac{1}{2}\mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| \geq 1/2\right].$$

Thus, the results in this section subsume those in Section 3-4.

Note that the class of read-once real formulas also includes the class of Boolean formulas which have been corrupted with the kind of random misclassification noise described in Section 3-5. This is because such noise can be simulated by a single, output-level gate $\text{LIN}_{\eta_0, 1-\eta_0-\eta_1}$: on input 0, this gate outputs η_0, and on input 1, it outputs $1 - \eta_1$. Thus, η_0 and η_1 are the respective probabilities that an input of 0 or 1 is "misclassified" or flipped by this gate.

In general, we can regard gates LIN_{zw} as describing the behavior of a "noisy" or randomized Boolean gate which on input 0 outputs 1 with probability z, and on input 1 outputs 1 with probability $z + w$. Clearly, if the input to such a randomized gate is 1 with probability p, then the gate outputs 1 with probability

$$(1 - p)z + p(z + w) = z + wp = \text{LIN}_{zw}(p).$$

Thus, the probabilistic behavior of a formula constructed with such randomized gates is described by the p-concept obtained by replacing each randomized gate by a LIN_{zw} gate, for an appropriate choice of z and w. (This can be proved rigorously, for instance, using a straightforward induction argument on the depth of the formula.)

Thus, our result can be viewed as a demonstration of the learnability of read-once Boolean formulas with large doses of noise sprinkled throughout the formulas. Such noise may affect the formula's output ("misclassification noise"), the inputs ("attribute noise") or it might affect the output of every gate of the formula.

3-7.1   Overview of the learning algorithm

Our learning procedure uses many of the ideas and techniques developed in the preceding sections. The algorithm operates in three stages. In Stage I, we estimate the probability p_i that each variable x_i is set to 1. Any "sticky" variables for which this probability is too close to 0 or 1 will be disregarded in later stages. We also determine the "influence" of each variable x_i: roughly speaking, this is the probability that the target formula's value changes significantly if x_i's setting is flipped. Variables which have very little influence are also considered irrelevant in later stages, similar to the manner in which insignificant (deeply occurring) variables are ignored in the algorithm of Section 3-6. Not surprisingly, sticky and uninfluential variables can be ignored without introducing much error.

In Stage II, we construct an approximation of the target formula's topology or skeleton structure. By the *skeleton* of a real formula, we refer to the topological structure of the formula, i.e., the formula stripped of the z, w-values on the LIN gates. For instance, the formula in (7.2) has skeleton:

$$\text{LIN}(\text{MUL}(\text{LIN}(x_1), \text{LIN}(x_2))).$$

In Stage II, we infer a skeleton σ which we show approximates the target formula in the sense that σ is the skeleton of some formula which is a good approximation (say, in the sense of (7.1)) of the target formula.

Finally, in Stage III, we approximate the z, w values of the LIN gates of the skeleton inferred in Stage II.

3-7.2   Some preliminary facts

Let f be the target formula, and let D be the target distribution. In what follows, all expectations are taken with respect to distribution D. For a function g, we also sometimes write $\bar{g}$ to denote its expectation:

$$\bar{g} = \mathbf{E}[g] = \mathbf{E}_{x \in D}[g(x)].$$

We will be interested in the partial derivatives of f, which turn out to be easily computed and their expectations easily approximated by sampling. We say that a gate λ is an *uncle* of variable x_i if λ is an immediate input to a MUL gate fed by x_i, but λ is not itself fed by x_i.

Lemma 7.1 *Let x_i be a relevant variable of f. Let $\{\text{LIN}_{z_j, w_j}\}$ be the sequence of LIN gates fed by x_i, and let $\{g_j\}$ be the sequence of subformulas subsumed by uncles of x_i. Then*

$$\frac{\partial f}{\partial x_i} = \prod_j w_j \cdot \prod_j g_j.$$

The same holds if x_i is replaced by a subformula of f.

Proof: By induction on the depth of f. If the depth is zero, then $f = x_i$, and the lemma holds. (The "empty" product is 1.)

Otherwise, if the output gate of f is a LIN_{zw} gate, then $f = \text{LIN}_{zw}(f') = z + w f_0$, for some subformula f'. (The notation f' is potentially confusing since f' sometimes denotes the derivative of f. However, here and throughout this chapter, f' simply denotes a function that, in general, is unrelated to any derivative of f.) Thus, $\partial f / \partial x_i = w \cdot (\partial f' / \partial x_i)$ and the lemma holds by inductive hypothesis. If the output gate is a MUL gate, then $f = f'g$, for some subformulas f' and g. Variable x_i is relevant to only, say, f'. Thus, g is a subformula subsumed by an uncle of x_i, and since x_i is not relevant to g, we have $\partial f / \partial x_i = g \cdot (\partial f' / \partial x_i)$. Thus, the lemma holds again in this case by inductive hypothesis.

Regarding a subformula g of f as a meta-variable, it can be seen that this same argument holds if x_i is replaced by subformula g. ∎

Since each g_j in this lemma is itself a real formula, g_j has range in $[0, 1]$. Also, $|w_j| \leq 1$ for each w_j. Thus, it follows from Lemma 7.1 that $|\partial f / \partial x_i| \leq 1$, and that the sign of $\partial f / \partial x_i$ is determined by the w_j's. Thus, $\partial f / \partial x_i$ is either a nonpositive or nonnegative function.

Note that the formula f could be "multiplied out" to give a polynomial over the variable set. This polynomial is linear in each variable x_i. This follows, for instance, from Lemma 7.1 since x_i is not relevant to any of the functions g_j, and so $\partial^2 f / \partial x_i^2 = 0$. Thus, we can write

$$f = u + v x_i$$

for some functions u and v to which x_i is not relevant. We call u and v the *decomposition* of f in terms of x_i. In the same manner, f can be decomposed in terms of any subformula g, and so can be written $f = u + vg$ for some functions u and v that do not contain any variable relevant to g.

Clearly, if $f = u + v x_i$, then

$$\frac{\partial f}{\partial x_i} = v = (f | x_i \leftarrow 1) - (f | x_i \leftarrow 0). \tag{7.3}$$

Since, as noted above, $\partial f / \partial x_i$ is either nonpositive or nonnegative on all inputs,

$$|\mathbf{E}\left[\partial f / \partial x_i\right]| = \mathbf{E}\left[|\partial f / \partial x_i|\right] = \mathbf{E}\left[|(f | x_i \leftarrow 1) - (f | x_i \leftarrow 0)|\right].$$

This latter expression is a natural measure of the *influence* of x_i, the degree to which x_i's value affects the value of f.

Also, equation (7.3) implies that

$$\mathbf{E}\left[\partial f / \partial x_i\right] = \mathbf{E}\left[f | x_i \leftarrow 1\right] - \mathbf{E}\left[f | x_i \leftarrow 0\right]. \tag{7.4}$$

Note that the expressions on the right can be easily approximated by simply estimating the probability that a positive example is received when x_i is hardwired to 0 or 1 (assuming x_i is not sticky). Thus, the expected value of $\partial f/\partial x_i$ can be easily estimated as the difference of these estimates.

In Stages II and III, we will also be interested in the expected value of the the second partial derivatives; these values turn out to be quite useful to our algorithm. If $i \neq j$, then since f is linear in every variable, f can be written as

$$f = u_0 + u_1 x_i + u_2 x_j + u_3 x_i x_j$$

for some functions u_0, u_1, u_2 and u_3 to which x_i and x_j are irrelevant. Then it is easily verified that

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = u_3 = (f|x_i \leftarrow 1, x_j \leftarrow 1) - (f|x_i \leftarrow 1, x_j \leftarrow 0)$$
$$-(f|x_i \leftarrow 0, x_j \leftarrow 1) + (f|x_i \leftarrow 0, x_j \leftarrow 0). \tag{7.5}$$

Since, as before, the expectation of each expression on the right can be estimated by sampling on filtered distributions, we can obtain a good estimate of $\mathbf{E}[\partial^2 f/\partial x_i \partial x_j]$.

Note that Lemma 7.1 shows that either $\partial f/\partial x_i$ or its negation is a read-once real formula. Thus, in either case, the lemma implies that the second partial derivative $\partial^2 f/\partial x_i \partial x_j$ has many of the properties of the first derivative described above: in particular, its magnitude is bounded by 1, and it is nonnegative or nonpositive on all inputs.

3-7.3   Stage I: Eliminating sticky and uninfluential variables

As described above, in Stage I, sticky and uninfluential variables are eliminated. Our algorithm begins by estimating the probability p_i that each variable x_i is set to 1 under the target distribution. Applying Chernoff bounds (Lemma 2-3.6), we see that a polynomial-size sample suffices to obtain estimates $\hat{p}_i$ such that, with probability at least $1 - \delta/4$, every estimate p_i is such that $|p_i - \hat{p}_i| \leq \epsilon/12n$. If $\hat{p}_i \leq \epsilon/4n$ or $\hat{p}_i \geq 1 - \epsilon/4n$, then we say that x_i is *sticky*. In this case, assuming the accuracy of our estimates, either $p_i \leq \epsilon/3n$ or $1 - p_i \leq \epsilon/3n$. Sticky variables are ignored in later stages of the algorithm.

If x_i is not sticky, then $\epsilon/6n \leq p_i \leq 1 - \epsilon/6n$. In this case, a good estimate of $\mathbf{E}[f|x_i \leftarrow b]$ can be obtained for $b \in \{0, 1\}$. We require that these estimates have accuracy $\epsilon^8/2(9n)^{10}$. (This high degree of accuracy is necessary for later stages of the algorithm.) With probability at least $1 - \delta/4$, such estimates can be obtained for all unsticky variables using a polynomial-size sample (again by applying Chernoff bounds).

These estimates can in turn be used to obtain estimates $\hat{B}_i$ of the expected value of $B_i =$

$\partial f/\partial x_i$ using equation (7.4). We then have $\left|\bar{B}_i - \hat{B}_i\right| \leq \epsilon^8/(9n)^{10}$. If $\left|\hat{B}_i\right| \geq \epsilon/4n$, we say that x_i is *influential*; in this case, $\left|\bar{B}_i\right| \geq \epsilon/5n$. In later stages, uninfluential variables are ignored; for these variables, $\left|\bar{B}_i\right| \leq \epsilon/3n$.

We show next that sticky and uninfluential variables can be ignored without introducing much error. Let f' be the read-once real formula obtained from f by replacing some variable x_i by the constant p_i (or, equivalently, since this is not technically a real formula, by $\text{LIN}_{p_i,0}(x_i)$). Formula f' is just the p-concept obtained by regarding x_i as a hidden variable — if we ignore x_i's value, then f' describes the probability that an assignment to the other variables is labeled 1. Note that $\bar{g} = \bar{g}'$ for every subformula g of f and its corresponding subformula g' of f'; this can be proved by an easy induction argument on the depth of g. Applying Lemma 7.1, this shows in particular that the influence of any other variable x_j is the same in f and f'.

Let u, v be the decomposition of f in terms of x_i so that $f = u + vx_i$. Then $f' = u + vp_i$, and so

$$f - f' = v(x_i - p_i) = (\partial f/\partial x_i)(x_i - p_i).$$

Thus, by independence,

$$\mathbf{E}\left[|f - f'|\right] = \mathbf{E}\left[|\partial f/\partial x_i|\right] \cdot \mathbf{E}\left[|x_i - p_i|\right] = \left|\bar{B}_i\right| \cdot 2p_i(1 - p_i).$$

Note that if x_i is sticky or uninfluential, then this latter expression is at most $2\epsilon/3n$.

Suppose $x_1, \ldots, x_s$ are the variables of f which are either sticky or uninfluential. Let $f_0 = f$, and let f_i be obtained from f_{i-1} by replacing x_i with the constant p_i for $1 \leq i \leq s$, and let $f_{\text{I}} = f_s$. Then

$$\begin{aligned}
\mathbf{E}\left[|f - f_{\text{I}}|\right] &\leq \sum_{i=1}^{s} \mathbf{E}\left[|f_i - f_{i-1}|\right] \\
&\leq 2s\epsilon/3n \leq 2\epsilon/3,
\end{aligned}$$

since, by the preceding argument, $\mathbf{E}\left[|f_i - f_{i-1}|\right] \leq 2\epsilon/3n$.

As in Section 3-6, we henceforth regard f_{I} as the target formula. Sampling according to f_{I} is achieved by simply ignoring the variables eliminated from f. In the later stages, it is shown how to find a hypothesis $\hat{f}$ such that $\mathbf{E}\left[\left|\hat{f} - f_{\text{I}}\right|\right] \leq \epsilon/3$, and thus $\mathbf{E}\left[\left|\hat{f} - f\right|\right] \leq \epsilon$.

Note that we can easily handle at this point the special case that all the variables of f are eliminated. For this case, f_{I} simply computes some constant function p. Since $\bar{f} = \bar{f}_{\text{I}} = p$, we can with high probability obtain an estimate $\hat{p}$ of p so that $|p - \hat{p}| \leq \epsilon/3$. Letting $\hat{f} = \hat{p}$ be our hypothesis, we have that $\mathbf{E}\left[\left|\hat{f} - f_{\text{I}}\right|\right] = |\hat{p} - p|$, and thus $\mathbf{E}\left[\left|f - \hat{f}\right|\right] \leq \epsilon$ as desired.

3-7.4   Stage II: Inferring the formula's skeleton

Based on the results of Stage I, we can assume henceforth that none of the variables of f are either sticky or uninfluential. Thus, in this section, f actually refers to the formula f_I of the previous section, and all of the variables discussed are assumed to be neither sticky nor uninfluential.

We show in this section how an approximation of the skeletal structure of f can be obtained. Specifically, we show how a structure σ can be inferred which is the skeleton of some formula f_{II} for which $|f - f_{II}|$ is very small on all inputs. In Stage III, we will see how a formula very close to f_{II} can be inferred from σ and other statistical information.

Note that the functional composition of two LIN gates is a LIN gate, and that $LIN_{0,1}$ is the identity function. Thus, without loss of generality, we assume the gates of f occur in alternating layers of MUL and LIN gates, the output gate being a LIN gate, and every variable an input to a LIN gate. Thus, the topology or skeleton of f is entirely determined by the tree structure of f with respect to the MUL gates. Therefore, in reconstructing f's skeleton, we will be quite interested in determining which MUL gates are fed by which other MUL gates, and in particular, which of two MUL gates occurs deeper in the formula.

Our algorithm uses two tests for determining which of two MUL gates is deeper. After describing the two tests, we show how a skeleton can be constructed from the results of these tests.

To simplify notation, we let $\Gamma_{ij} = \Gamma(x_i, x_j)$, and we write g_{ij} to denote the subformula subsumed by Γ_{ij}.

For any three variables x_i, x_j and x_k, we must have that two of the gates Γ_{ij}, Γ_{ik} and Γ_{jk} are actually the same, and that the remaining gate is the deepest of the three. For instance, if Γ_{ij} is the deepest gate, then it feeds $\Gamma_{ik} = \Gamma_{jk}$. Our purpose, initially, is to determine which of these gates is deepest. This will be determined from the expected values of the first and second partial derivatives of f.

Recall that good estimates of $\bar{B}_i = \mathbf{E}[\partial f/\partial x_i]$ were obtained in Stage I. Let $A_{ij} = \partial^2 f/\partial x_i \partial x_j$. From equation (7.5), it follows that good estimates $\hat{A}_{ij}$ of $\bar{A}_{ij}$ can also be obtained. Specifically, for each pair of variables, and each $b_1, b_2 \in \{0, 1\}$, we estimate $\mathbf{E}[f|x_i \leftarrow b_1, x_j \leftarrow b_2]$ to within accuracy $\epsilon^8/4 \cdot (9n)^{10}$. Such accuracy can be achieved (for all unsticky variables) with probability at least $1 - \delta/4$ using a polynomial-size sample. Estimates of $\bar{A}_{ij}$ can then be derived using equation (7.5) since

$$
\begin{aligned}
\bar{A}_{ij} &= \mathbf{E}\left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right] \\
&= \mathbf{E}[f|x_i \leftarrow 1, x_j \leftarrow 1] - \mathbf{E}[f|x_i \leftarrow 1, x_j \leftarrow 0] - \mathbf{E}[f|x_i \leftarrow 0, x_j \leftarrow 1] + \mathbf{E}[f|x_i \leftarrow 0, x_j \leftarrow 0].
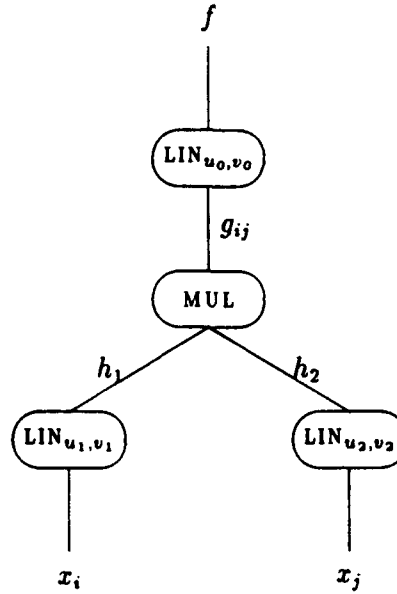\end{aligned}
$$

Figure 3: The decomposition of f used in Lemma 7.2.

We assume henceforth that all of the estimates have the desired accuracy. Then estimates of $\bar{A}_{ij}$ derived in this manner are such that $\left|\hat{A}_{ij} - \bar{A}_{ij}\right| \leq \epsilon^8/(9n)^{10}$. It will also be convenient to assume that each $\hat{B}_i$ and $\hat{A}_{ij}$ is in the range $[-1, 1]$; since $\bar{B}_i$ and $\bar{A}_{ij}$ are known to be in this range, we make this assumption without loss of generality. ("Clamping" estimates in this range can only improve their accuracy.)

The sign test

Our first test for determining which of two gates is deeper in f is called the *sign test*, and it is based on our ability to determine the sign of certain partial derivatives as described below.

Specifically, for variables x_i and x_j, we show below that it is possible to determine the sign of $\mathbf{E}\left[\partial f/\partial g_{ij}\right]$, which we denote by c_{ij}. To see that this might be useful, note that if $\Gamma_{ik} = \Gamma_{jk}$ then certainly $g_{ik} = g_{jk}$, and so $c_{ik} = c_{jk}$. Thus, if for some triple x_i, x_j and x_k we find that $c_{ij} \neq c_{ik} = c_{jk}$, then we can conclude that Γ_{ij} occurs deeper in f than $\Gamma_{ik} = \Gamma_{jk}$. (Recall that exactly two of the gates Γ_{ij}, Γ_{ik} and Γ_{jk} must be equal to one another; since $c_{ij} \neq c_{ik} = c_{jk}$, this is the only possibility.) This is the essence of the sign test.

Lemma 7.2 *Let x_i and x_j be distinct variables. Then*

$$sign(\mathbf{E}\left[\partial f/\partial g_{ij}\right]) = sign(\hat{B}_i \cdot \hat{B}_j \cdot \hat{A}_{ij}).$$

Proof: Since pairs of variables can only meet at MUL gates, we can write g_{ij} as a product of its

inputs, $g_{ij} = h_1 h_2$, where h_1 and h_2 are subformulas containing x_i and x_j, respectively. We can decompose h_1 and h_2 in terms of x_i and x_j, and so can write $h_1 = u_1 + v_1 x_i$ and $h_2 = u_2 + v_2 x_j$. We can also decompose f in terms of g_{ij}: $f = u_0 + v_0 g_{ij}$. (This decomposition is summarized in Figure 3.)

Thus, we can easily compute that:

$$B_i = v_0 v_1 h_2$$

$$B_j = v_0 h_1 v_2$$

$$A_{ij} = v_0 v_1 v_2.$$

We have that $|v_0| \leq 1$ by Lemma 7.1, and h_1 and h_2, being subformulas, are in the range $[0, 1]$. Thus,

$$\left| \bar{A}_{ij} \right| = \left| \bar{v}_0 \bar{v}_1 \bar{v}_2 \right| \geq \left| \bar{v}_0^2 \bar{v}_1 \bar{v}_2 \bar{h}_1 \bar{h}_2 \right| = \left| \bar{B}_i \right| \left| \bar{B}_j \right|.$$

This last expression is at least $(\epsilon/5n)^2$ since x_i and x_j are influential. Thus, $sign(\bar{A}_{ij}) = sign(\hat{A}_{ij})$. Likewise, the signs of $\bar{B}_i$ and $\bar{B}_j$ can be determined from their estimates.

From the expressions above, and since h_1 and h_2 are nonnegative, it is clear that $sign(\bar{v}_0) = sign(\bar{B}_i \cdot \bar{B}_j \cdot \bar{A}_{ij})$. Since $\bar{v}_0 = \mathbf{E}\left[\partial f / \partial g_{ij}\right]$, this proves the lemma.   ∎

Thus, as described above, our algorithm computes $c_{ij} = sign(\mathbf{E}\left[\partial f / \partial g_{ij}\right])$ for each pair of variables x_i and x_j. Then, for each triple x_i, x_j and x_k, the algorithm tests whether $c_{ij} \neq c_{ik} = c_{jk}$. If it finds that this is the case, then the algorithm can correctly conclude that $\Gamma_{ik} = \Gamma_{jk}$, and therefore Γ_{ij} occurs deeper than this gate.

The results of all these sign tests are organized in a directed graph G_s. The nodes of G_s are unordered pairs $\{i, j\}$, for $i \neq j$. For each ordered triple of distinct indices i, j, k, our algorithm tests if $c_{ij} \neq c_{ik} = c_{jk}$. If this is the case, an edge is directed in G_s from $\{i, j\}$ to $\{i, k\}$. Thus, as argued above, an edge is added to G_s in this fashion only if Γ_{ij} is deeper in f than Γ_{ik}. Moreover, by transitivity, a path from $\{i, j\}$ to $\{i', j'\}$ implies that Γ_{ij} is deeper than $\Gamma_{i'j'}$.

Note that the sign test is a one-sided test in the sense that if $c_{ij} = c_{ik} = c_{jk}$ then nothing can be concluded about the relative depth of Γ_{ij} and Γ_{ik}. However, our next lemmas give conditions under which the sign test and its graph G_s are guaranteed to give such depth information.

Lemma 7.3 *Let x_i, x_j and x_k be distinct variables of f, and assume Γ_{ij} is deeper in f than $\Gamma_{ik} = \Gamma_{jk}$. Then $c_{ij} \neq c_{ik}$ if and only if $\mathbf{E}\left[\partial g_{ik}/\partial g_{ij}\right] < 0$.*

Proof: This follows immediately from the chain rule

$$\frac{\partial f}{\partial g_{ij}} = \frac{\partial f}{\partial g_{ik}} \cdot \frac{\partial g_{ik}}{\partial g_{ij}},$$

which implies by independence that $c_{ij} = c_{ik} \cdot sign(\mathbf{E}\left[\partial g_{ik}/\partial g_{ij}\right])$.   ∎

Lemma 7.4 *Let x_i, x_j and x_k be distinct variables of f, and assume Γ_{ij} is deeper in f than $\Gamma_{ik} = \Gamma_{jk}$. Assume also that the path in f from Γ_{ij} to Γ_{ik} includes a gate LIN_{zw} with $w < 0$. Then there exists a path in G_* from $\{i,j\}$ to $\{i,k\}$.*

Proof: Let $\{\lambda_r\}_{r=1}^t$ be the sequence of LIN gates on the path from Γ_{ij} to Γ_{ik}, and suppose that each λ_r computes the function $\text{LIN}_{z_r w_r}$.

By Lemma 7.1, the sign of $\partial g_{ik}/\partial g_{ij}$ is given by the sign of the product $\prod w_r$. Thus, if $\prod w_r < 0$, then $\partial g_{ik}/\partial g_{ij}$ is nonpositive, and so by Lemma 7.2, an edge is directed in G_* from $\{i,j\}$ to $\{i,k\}$, and the lemma holds in this case.

Otherwise, $\prod w_r \geq 0$. Note that $\prod w_r$ is nonzero since if it were zero, then by Lemma 7.1, $\partial f/\partial x_i$ would be zero (since the path from Γ_{ij} to Γ_{ik} is a subpath of the path from x_i to the output), contradicting the assumption that x_i is influential.

Thus, $\prod w_r > 0$. Since we assume some $w_r < 0$, there is some s such that $\prod_{r=1}^s w_r < 0$ and $\prod_{r=s+1}^t w_r < 0$. Since, by assumption, no LIN gate is the input to another LIN gate, there must be a MUL gate separating λ_r and λ_{r+1}; since this MUL gate is fed by x_i, it can be written $\Gamma_{i\ell}$ for some variable x_ℓ. Applying the first part of this argument twice (the case that $\prod w_r < 0$), it follows that there must exist an edge in G_* from $\{i,j\}$ to $\{i,\ell\}$, and another edge from $\{i,\ell\}$ to $\{i,k\}$. This proves the lemma. ∎

Thus, the sign test succeeds in determining which of two gates Γ_{ij} and Γ_{ik} occurs deeper in f whenever the path from one gate to the other includes a LIN_{zw} gate with $w < 0$. We describe next another test for handling all other situations.

The product test

The second test is called the *product test*, and it has a flavor similar to that of the sign test. We will be interested in the quantities $d_{ijk} = \bar{B}_i \cdot \bar{A}_{jk}$. Note that $d_{ijk} = d_{ikj}$ always. We will see that if Γ_{ij} occurs deeper than Γ_{ik}, then $d_{ijk} = d_{jik}$, but that if Γ_{ik} occurs deeper than Γ_{ij} then d_{jik} is apt to differ significantly from d_{ijk}. Thus, the values of d_{ijk} will give a second method for determining which of the three gates Γ_{ij}, Γ_{ik} and Γ_{jk} occurs deepest in f.

Lemma 7.5 *Let x_i, x_j and x_k be distinct variables, and assume Γ_{ij} occurs deeper in f than $\Gamma_{ik} = \Gamma_{jk}$. Then $d_{ijk} = d_{jik}$.*

Proof: We can decompose f in terms of g_{ik} as $f = u_0 + v_0 g_{ik}$, so that none of the variables relevant to g_{ik} are relevant to the functions u_0 and v_0. Since Γ_{ik} and Γ_{ij} are MUL gates, we can express the function g_{ik} in terms of the subformulas that it subsumes: $g_{ik} = yy_3$. Gate Γ_{ij} occurs in one of these subformulas, say y, and x_k occurs in the other y_3. We can decompose y and y_3 in terms of g_{ij} and x_k, respectively, so that $y = u + v g_{ij}$, and $y_3 = u_3 + v_3 x_k$. Similarly, Γ_{ij} is a MUL gate, so we can write $g_{ij} = y_1 y_2$ where y_1 and y_2 contain x_i and x_j, respectively.
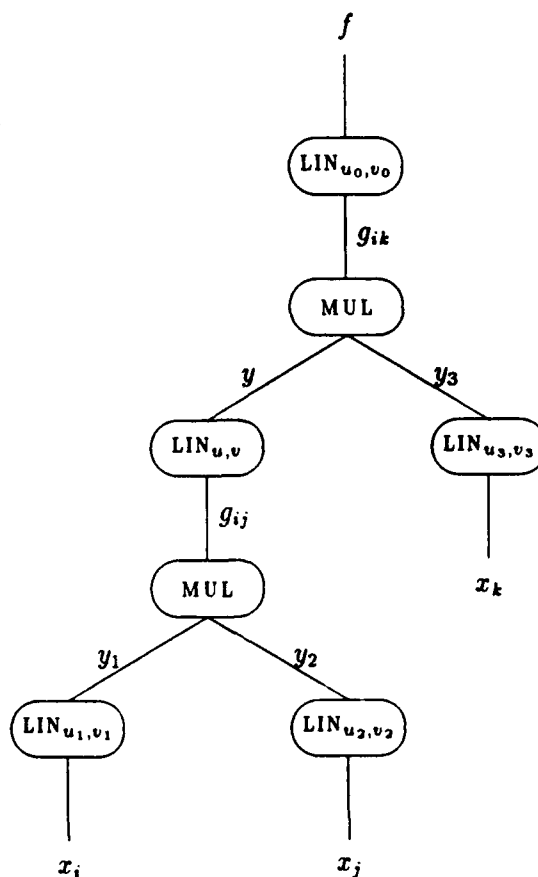
Figure 4: The decomposition of f used in Lemmas 7.5 and 7.6.

Decomposing y_1 and y_2, we can write $y_1 = u_1 + v_1 x_i$ and $y_2 = u_2 + v_2 x_j$. (This decomposition of f is summarized in Figure 4.)

By a direct computation, we have that:

$$
\begin{aligned}
B_i &= v v_0 v_1 y_2 y_3 \\
B_j &= v v_0 y_1 v_2 y_3 \\
B_k &= y v_0 v_3 \\
A_{ij} &= v v_0 v_1 v_2 y_3 \\
A_{jk} &= v v_0 y_1 v_2 v_3 \\
A_{ik} &= v v_0 v_1 y_2 v_3.
\end{aligned}
$$

Thus, by independence,

$$
d_{ijk} = \bar{v}^2 \bar{v}_0^2 \bar{v}_1 \bar{v}_2 \bar{v}_3 \bar{y}_1 \bar{y}_2 \bar{y}_3 = d_{jik}.
$$

■

As defined in this proof, we call the function u the *offset* of g_{ik} with respect to g_{ij}, and we denote it offset(g_{ik}, g_{ij}).

Lemma 7.6 *Let x_i, x_j and x_k be distinct variables, and assume Γ_{ij} occurs deeper in f than $\Gamma_{ik} = \Gamma_{jk}$. Then $|d_{ijk} - d_{kij}| \geq (\epsilon/5n)^3 \cdot |\mathbf{E}\,[\mathrm{offset}(g_{ik}, g_{ij})]|$.*

Proof: With the same set-up as in the preceding lemma, we have that

$$d_{kij} = \bar{v}\bar{y}\bar{v}_0^2\bar{v}_1\bar{v}_2\bar{v}_3\bar{y}_3.$$

Thus, since $y = u + vy_1y_2$,

$$d_{kij} - d_{ijk} = \alpha \cdot (\bar{y} - \bar{v}\bar{y}_1\bar{y}_2) = \alpha\bar{u},$$

where $\alpha = \bar{v}\bar{v}_0^2\bar{v}_1\bar{v}_2\bar{v}_3\bar{y}_3$. Note that $|\alpha| = |\bar{B}_k \cdot \bar{A}_{ij}/\bar{y}| \geq (\epsilon/5n)^3$ since $|\bar{y}| \leq 1$, and since all of the variables are influential. (It was shown in the proof of Lemma 7.2 that $|\bar{A}_{ij}| \geq |\bar{B}_i| \cdot |\bar{B}_j|$.) This implies the lemma.   ■

We use $\hat{d}_{ijk} = \hat{B}_i \cdot \hat{A}_{jk}$ to estimate d_{ijk}. Then

$$\begin{aligned}\left|\hat{d}_{ijk} - d_{ijk}\right| &\leq \left|\hat{B}_i\right|\left|\hat{A}_{jk} - \bar{A}_{jk}\right| + |\bar{A}_{jk}|\left|\hat{B}_i - \bar{B}_i\right| \\ &\leq 2\epsilon^8/(9n)^{10}\end{aligned}$$

using the fact that $\left|\hat{B}_i\right|$ and $|\bar{A}_{jk}|$ are both bounded by 1. As before for the sign test, we maintain a graph G_p with vertices $\{i, j\}$ for $i \neq j$. Edges are directed from $\{i, j\}$ to $\{i, k\}$ and also from $\{j, k\}$ to $\{i, k\}$ for all ordered triples i, j, k which satisfy $\left|\hat{d}_{ijk} - \hat{d}_{jik}\right| \leq 4\epsilon^8/(9n)^{10}$. Thus, by Lemma 7.5, if Γ_{ij} is deeper than $\Gamma_{ik} = \Gamma_{jk}$, then $d_{ijk} = d_{jik}$ and so edges are directed from $\{i, j\}$ to $\{i, k\}$ and from $\{j, k\}$ to $\{i, k\}$. However, the product test is one sided (though in a different way than the sign test): it may happen that such edges are added even if Γ_{ij} is above Γ_{ik}. Nevertheless, by Lemma 7.6, this can only be the case if $|\mathbf{E}\,[\mathrm{offset}(g_{ik}, g_{ij})]|$ is quite small.

Constructing the skeleton from G_s and G_p

Finally, we are ready to show how an "approximate" skeleton of f can be computed from the graphs G_s and G_p. First, to reiterate what was pointed out above, an edge in G_s from $\{i, j\}$ to $\{i, k\}$ indicates that Γ_{ij} *must* be deeper than Γ_{ik}; however an edge in G_p between these vertices indicates only that Γ_{ij} *may* be deeper than or equal to Γ_{ik} (although it might not be). On the other hand, if Γ_{ij} is deeper than Γ_{ik}, or if $\Gamma_{ij} = \Gamma_{ik}$, then there *must* be an edge in G_p from $\{i, j\}$ to $\{i, k\}$, but possibly not in G_s.

We can combine G_p and G_s into a single graph G_c on the same vertex set: From the comments above, it follows that the edge set of G_s is a subset of the edge set of G_p. The graph of G_c is obtained from G_p by deleting all edges from $\{i,j\}$ to $\{i,k\}$ which "contradict" G_s, i.e., for which there exists a path in G_s from $\{i,k\}$ to $\{i,j\}$. Such edges can be removed with impunity since the reverse path in G_s indicates that Γ_{ik} must be deeper than Γ_{ij}.

It is easily seen that if Γ_{ij} is deeper than Γ_{ik}, or if $\Gamma_{ij} = \Gamma_{ik}$ then G_c will contain an edge from $\{i,j\}$ to $\{i,k\}$ since the corresponding edge in G_p will not be deleted.

Thus, if there exists a path in G_c from $\{i,j\}$ to $\{k,\ell\}$, but no path in the reverse direction, then we can conclude that Γ_{ij} is deeper in f than $\Gamma_{k\ell}$. The problem arises when there exist paths connecting these vertices in both directions, that is, when $\{i,j\}$ and $\{k,\ell\}$ are in the same strongly connected component of G_c. We therefore need some way of dealing with these strongly connected components.

We say two vertices of G_c are *equivalent* (with respect to G_c) if they are in the same strongly connected component.

We say that a LIN gate of f is *trapped* if it is immediately fed by some MUL gate Γ_{ij}, and it immediately feeds another MUL gate $\Gamma_{k\ell}$ for some indices i, j, k, ℓ which are such that $\{i,j\}$ and $\{k,\ell\}$ are equivalent. In other words, the LIN gate is trapped if it is "surrounded" above and below by gates whose indices are in the same strongly connected component of G_c. We will see that if some gate LIN_{zw} is trapped, then its z-value must be quite small, and so we will be able to approximate such gates by a $\text{LIN}_{0,w}$ gate. As will be seen, the strongly connected components of G_c correspond to connected regions of f (where we view f as a graph whose vertices are the gates, and whose edges are the "wires" connecting the gates); thus we will be able to approximate these connected regions by simple MUL gates.

Below, we let $\theta = (8 \cdot 5^4/9^{10})\epsilon^4/n^6$.

Lemma 7.7 *Suppose λ is a trapped LIN_{zw} gate of f. Then $w \geq 0$ and $z \leq \theta$.*

Proof: Since λ is trapped, it is immediately fed by some MUL gate Γ_{ij}, and it immediately feeds MUL gate $\Gamma_{k\ell}$ where $\{i,j\}$ and $\{k,\ell\}$ are equivalent. Let T be the set of vertices $\{i',j'\}$ for which $\Gamma_{i'j'}$ feeds Γ_{ij}, or $\Gamma_{i'j'} = \Gamma_{ij}$. Then $\{k,\ell\} \notin T$, since Γ_{ij} feeds $\Gamma_{k\ell}$. Since there exists a path from $\{k,\ell\}$ to $\{i,j\}$, there must be an edge directed from one vertex $\{i',k'\} \notin T$ to another $\{i',j'\} \in T$. Since $\Gamma_{i'j'}$ feeds or is equal to Γ_{ij}, the path in f from $\Gamma_{i'j'}$ to $\Gamma_{i'k'}$ must include gate λ.

We are interested in examining this path more closely. Let $h_0, h_1, \ldots, h_r$ be the sequence of subformulas subsumed by the MUL gates along this path. Thus, $h_0 = g_{i'j'}$, and $h_r = g_{i'k'}$. Further, since each consecutive pair of MUL gates is separated by some gate $\text{LIN}_{z_t w_t}$, we can write $h_t = y_t(z_t + w_t h_{t-1})$, where y_t is a subformula subsumed by an uncle of $g_{i'j'}$, for $1 \leq t \leq r$.

Note that no $w_t < 0$. If it were, then by Lemma 7.4, there would be a path from $\{i', j'\}$ to $\{i', k'\}$ in G_s, and so, by construction of G_c, there could not be an edge in G_c from $\{i', k'\}$ to $\{i', j'\}$. In particular, since λ is on the path from $\Gamma_{i',j'}$ to $\Gamma_{i'k'}$, this implies $w \geq 0$.

We can decompose the subformula $z_t + w_t h_{t-1}$ in terms of h_0 so that $z_t + w_t h_{t-1} = u_t + v_t h_0$. Thus, $h_t = y_t(u_t + v_t h_0)$. Note that $u_t = \text{offset}(h_t, h_0)$. Expanding, this implies for $t \geq 2$ that

$$
\begin{aligned}
h_t &= y_t(z_t + w_t h_{t-1}) \\
&= y_t(z_t + w_t y_{t-1}(u_{t-1} + v_{t-1}h_0)) \\
&= y_t(z_t + w_t y_{t-1}u_{t-1} + w_t y_{t-1}v_{t-1}h_0) \\
&= y_t(u_t + v_t h_0).
\end{aligned}
$$

Thus,

$$
u_t = z_t + w_t y_{t-1}u_{t-1}
$$

for $t \geq 2$. For $t = 1$, we have that $h_1 = y_1(u_1 + v_1 h_0) = y_1(z_1 + w_1 h_0)$ so $u_1 = z_1$. By a straightforward induction on r, it follows that

$$
\text{offset}(h_r, h_0) = u_r = \sum_{s=1}^{r}\left(z_s \cdot \prod_{t=s+1}^{r} w_t y_{t-1}\right).
$$

Since each $w_t \geq 0$, this sum is at least $z_s \prod_{t=s+1}^{r} w_t y_{t-1}$ for any s. Note that, by Lemma 7.1, $\prod_{t=s+1}^{r} w_t y_{t-1} \geq |\partial f/\partial x_{i'}|$ since the path in f from $x_{i'}$ to the output includes the path from $\Gamma_{i'j'}$ to $\Gamma_{i'k'}$. Thus, by Lemma 7.6 and our criterion for adding edges to G_p, and since $x_{i'}$ is influential,

$$
z_s \frac{\epsilon}{5n} \leq |\mathbf{E}\left[\text{offset}(h_r, h_0)\right]| \leq \left(\frac{5n}{\epsilon}\right)^3 \cdot \frac{8\epsilon^8}{(9n)^{10}}.
$$

Since $z = z_s$ for some s, this proves the lemma.   ∎

Let $f' = f_{\text{II}}$ be the real formula obtained from f by replacing all trapped gates LIN_{zw} by LIN_{0w}. Then f' is a good approximation of f:

Lemma 7.8 *Let f and $f' = f_{\text{II}}$ be as above. Then $|f(x) - f'(x)| \leq 2n\theta$ for all inputs x.*

Proof: We show by induction on the height of f that $|f - f'| \leq s\theta$ for all inputs, where s is the number of LIN gates occurring in f. We prove the lemma in a slightly stronger form for any f' obtained from f by replacing each LIN_{zw} gate by a $\text{LIN}_{z'w}$ gate for any z' satisfying $|z - z'| \leq \theta$; also, z' must be such that z' and $w + z'$ are in the range $[0, 1]$ so that f' is a real formula. Note that the f' of the hypothesis satisfies these conditions by Lemma 7.7.

If $f = x_i$, then $f' = x_i = f$, and the lemma holds.

If $f = \text{LIN}_{zw}(y)$, then $f' = \text{LIN}_{z'w}(y')$ for some z' and y' satisfying $|z - z'| \leq \theta$ and, by

Input: the graph G_c
Output: a skeleton that approximates f
Procedure:
1 $G \leftarrow G_c$
2 $F \leftarrow \{\text{LIN}(x_i) : 1 \le i \le n\}$
3 **repeat** while G is not empty
4   find a strongly connected component H of G with no incoming edges
5      $E \leftarrow \{\sigma \in F : \{i, j\} \in H \text{ and } x_i \in \text{rel}(\sigma)\}$
6      $F \leftarrow (F - E) \cup \{\text{LIN}(\text{MUL}_{\sigma \in E}\sigma)\}$
7      $G \leftarrow G - H$
8 **end**
9 **output** the only member of F

Figure 5: An algorithm for inferring a good skeleton from G_c.

inductive hypothesis, $|y - y'| \le (s - 1)\theta$. Then on all inputs,

$$
\begin{aligned}
|f - f'| &= |z - z' + w(y - y')| \\
&\le |z - z'| + |w||y - y'| \\
&\le s\theta
\end{aligned}
$$

since $|w| \le 1$.

If $f = \text{MUL}(y_1, y_2) = y_1 y_2$, where s_1 and s_2 LIN gates occur in y_1 and y_2, respectively, then $s_1 + s_2 = s$, and $f' = y_1' y_2'$ where $|y_i - y_i'| \le s_i \theta$ for $i = 1, 2$. Then on all inputs,

$$
\begin{aligned}
|f - f'| &= |y_1 y_2 - y_1' y_2'| \\
&= |y_1 y_2 - y_1 y_2' + y_1 y_2' - y_1' y_2'| \\
&\le |y_1||y_2 - y_2'| + |y_2'||y_1 - y_1'| \\
&\le s_2 \theta + s_1 \theta = s\theta.
\end{aligned}
$$

This completes the induction.

Since f has at most $2n$ LIN gates, this also proves the lemma.   ∎

Finally, we give an algorithm that infers the skeleton of a formula h that equals f'. (That is, h may differ syntactically from f', but it is functionally equivalent.) The algorithm is shown in Figure 5. The algorithm maintains a family of skeletons F. On each iteration of the loop, several of these skeletons are combined into one. We will show that only one skeleton remains in F upon termination. Also, although the MUL operator is technically binary, the multiple-input product used at line 6 can be replaced in the obvious manner by a tree of MUL gates. The set

rel(σ) for skeleton or subformula σ is the set of variables relevant to σ. Finally, $G - H$ at line 7 denotes the graph obtained from G by deleting all vertices in H, and any edges incident to these deleted vertices.

We say a condition holds *at all times* if it holds between each iteration of the main loop.

The algorithm clearly halts since some vertex of G is removed on each iteration.

Lemma 7.9 *At all times, if σ_1 and σ_2 are distinct members of F, then* rel(σ_1) $\cap$ rel(σ_2) = $\emptyset$. *Also,* $\bigcup_{\sigma \in F}$ rel(σ) = $\{x_1, \ldots, x_n\}$.

Proof: Using the fact that rel(LIN(MUL$_{\sigma \in E}\sigma$)) = $\bigcup_{\sigma \in E}$ rel(σ), this follows by an easy induction on the number of iterations of the main loop. ∎

Lemma 7.10 *At all times, if $\{i, j\}$ is not a vertex of G then $\{x_i, x_j\} \subset$ rel(σ) for some $\sigma \in F$.*

Proof: Note that $\{i, j\}$ is removed from G at line 7 only if x_i and x_j are relevant to skeletons in E. ∎

Lemma 7.11 *Upon termination of the main loop, $|F| = 1$.*

Proof: If, upon termination, F contained two skeletons σ_1 and σ_2, then each must have distinct relevant variables x_i and x_j, respectively, by Lemma 7.9. By Lemma 7.10, this implies $\{i, j\} \in G$, contradicting the fact that G is empty. ∎

Lemma 7.12 *At all times, if $\sigma \in F$ then σ is the skeleton of some formula which equals a subformula of f'.*

Proof: By induction on the number of iterations of the main loop. Initially, the lemma holds trivially.

Consider the state of the algorithm immediately before G is modified at line 7. Let $E = \{\sigma_1, \ldots, \sigma_r\}$ be as in the algorithm, and let $\sigma = $ LIN(MUL$_{i=1}^{r}\sigma_i$). Let $\{k, \ell\} \in H$ be such that $\Gamma_{k\ell}$ is a gate of minimal depth in the set $\{\Gamma_{ij} : \{i, j\} \in H\}$. (That is, $\Gamma_{k\ell}$ does not feed any gate in this set.)

Claim: rel(σ) = rel($g_{k\ell}$).

Proof of claim: Note first that, by definition of E,

$$\text{rel}(\sigma) = \{x_i : \{i, j\} \in H\}.$$

Thus, if $x_i \in$ rel(σ) then $\{i, j\} \in H$ for some j. Let T be the set

$$\{\{i', j'\} \in G : \Gamma_{i'j'} \text{ feeds } \Gamma_{k\ell} \text{ or } \Gamma_{i'j'} = \Gamma_{k\ell}\}.$$

We will show that $\{i,j\} \in T$; this implies that x_i is in the subformula subsumed by $g_{k\ell}$, and thus $x_i \in \text{rel}(g_{k\ell})$.

If $\{i,j\} \notin T$, then since $\{k,\ell\} \in T$ and since $\{i,j\}$ and $\{k,\ell\}$ are equivalent, there must be an edge in H from some node $\{i',j'\} \notin T$ to some other node $\{i',k'\} \in T$. Then $\Gamma_{i'k'}$ must feed $\Gamma_{i'j'}$, and $\Gamma_{k\ell}$ must be on the path in f from $\Gamma_{i'k'}$ to $\Gamma_{i'j'}$. Thus, $\Gamma_{i'k'}$ feeds (or is equal to) $\Gamma_{k\ell}$, which in turn feeds $\Gamma_{i'j'}$. Thus, there exist paths in G_c from $\{i',k'\}$ to $\{k,\ell\}$, and from $\{k,\ell\}$ to $\{i',j'\}$. Since an edge is directed from $\{i',j'\}$ to $\{i',k'\}$, this implies that $\{i',j'\}$ is in H. However, this contradicts the definition of $\{k,\ell\}$ since $\{k,\ell\}$ is deeper than $\{i',j'\}$.

Thus, $\text{rel}(\sigma) \subset \text{rel}(g_{k\ell})$.

Conversely, suppose $x_i \in \text{rel}(g_{k\ell})$. Then $\Gamma_{i\ell}$ feeds or is equal to $\Gamma_{k\ell}$. Thus, there is a path in G_c from $\{i,\ell\}$ to $\{k,\ell\}$. Since H is assumed to have no incoming edges, this implies that either $\{i,\ell\} \in H$ or $\{i,\ell\} \notin G$. If $\{i,\ell\} \notin G$, then by Lemma 7.10, $x_i \in \text{rel}(\sigma)$ since $x_\ell \in \text{rel}(\sigma)$. If $\{i,\ell\} \in H$, then $x_i \in \text{rel}(\sigma)$ by construction. Thus, $\text{rel}(\sigma) = \text{rel}(g_{k\ell})$, proving the claim.

The gate $\Gamma_{k\ell}$ in f' immediately feeds some LIN gate. Let h be the subformula of f' subsumed by this LIN gate. We will show that σ is the skeleton of a formula that equals h.

By inductive hypothesis, each σ_t is the skeleton of a formula equal to some subformula h_t of f'. The above claim implies that each h_t is in fact a subformula of h. Thus, h can be written as $h = g(h_1, \ldots, h_r)$ for some read-once real formula g. That is, g is what remains of h when each h_t is replaced by a meta-variable.

Consider a gate Γ_{ij} in h that does not belong to any of the subformulas h_t (and thus that remains in g). Since Γ_{ij} is in h, x_i and x_j are in $\text{rel}(\sigma)$, from the above claim. However, since Γ_{ij} is not in any of the subformulas h_t, x_i and x_j must be in different subformulas; thus, $\{i,j\} \in G$ by Lemma 7.10. Since Γ_{ij} feeds or is equal to $\Gamma_{k\ell}$, there exists a path in G_c from $\{i,j\}$ to $\{k,\ell\}$. Thus, $\{i,j\} \in H$.

This implies that every internal LIN gate of g is trapped since $\{i,j\} \in H$ for every MUL gate Γ_{ij} of g. Thus, if LIN_{zw} is an internal gate of g then $z = 0$ by construction of f', and the LIN gate is simply multiplying its input by a constant. Since

$$\text{MUL}\big(\text{LIN}_{0,w_1}(y_1), \text{LIN}_{0,w_2}(y_2)\big) = \text{LIN}_{0,w_1w_2}(\text{MUL}(y_1, y_2)),$$

this implies that all of the LIN gates of g can be "pulled to the top." In other words, we can write

$$h = \text{LIN}_{zw}\big(\text{MUL}_{t=1}^r h_t\big)$$

for an appropriate choice of z and w. This completes the lemma. ∎

Combining Lemmas 7.9, 7.11, and 7.12, it follows immediately that the algorithm of Figure 5 outputs a skeleton of a formula equal to f'. The algorithm clearly runs in polynomial time.

3-7.5   Stage III: Inferring the skeleton's z, w values

In the final stage, our algorithm "fills in" the missing z, w values of the skeleton inferred in Stage II. Recall that this skeleton σ was for a function f_{II} which closely approximates the target f. We would like to view f_{II} as the target since we have its skeleton to work with. The problem is that we have no way of sampling according to f_{II} — we can only sample using the target we have been provided with, and f_{II} may differ slightly from the target.

However, it turns out that this is not a problem since f and f_{II} are so close to one another. We showed in Lemma 7.8 that $|f(x) - f_{\text{II}}(x)| \le 2n\theta$ on all inputs x. Thus, for *any* distribution on the inputs (such as the filtered distributions used for estimating the expected values of the partial derivatives), the expected value of f is within $2n\theta$ of the expected value of f_{II}. Thus, we can view f_{II} henceforth as the target concept f, taking into account the fact that all of our statistical estimates are off by an additional factor of $2n\theta$. In particular, for all i, j,

$$\left| \hat{B}_i - \bar{B}_i \right| \le \epsilon^8/(9n)^{10} + 4n\theta \le a\epsilon^4/n^5$$
$$\left| \hat{A}_{ij} - \bar{A}_{ij} \right| \le \epsilon^8/(9n)^{10} + 8n\theta \le a\epsilon^4/n^5$$

where

$$a = (64 \cdot 5^4 + 1)/9^{10}.$$

Note that B_i now refers to $\partial f/\partial x_i$ for $f = f_{\text{II}}$ but the estimates $\hat{B}_i$ are unchanged; likewise for A_{ij}. Thus, for this new target formula, influential variables are such that $\bar{B}_i \ge \epsilon/4n - a\epsilon^4/n^5 \ge \epsilon/5n$.

Our algorithm uses these values to estimate the z, w values of the LIN gates of f. The algorithm computes the z, w values from the bottom up: the algorithm visits each gate of the skeleton, starting with those nearest the input level. No gate is visited until all those that feed it have been visited. When some gate λ is visited which subsumes subformula g, our algorithm computes a formula that approximates not the function g itself, but rather the function $g/\bar{g}$. (Apparently, $g/\bar{g}$ is easier to approximate than g.) This is done for all gates except the output-level gate where an approximation of the function f itself is computed.

Although not the case for the skeleton computed in Stage II, we nevertheless assume without loss of generality that the gates of σ occur in alternating layers of MUL/LIN gates, with a LIN gate at the output, and every variable an input to a LIN gate.

Suppose that our algorithm is visiting some gate of σ, and assume that the corresponding gate λ of f subsumes subformula g. There are five cases to consider:

1. λ is a MUL gate, and thus λ is immediately fed by a LIN gate, and immediately feeds a LIN gate;

2. λ is a LIN gate immediately fed by a variable, and λ is not the output gate;

3. λ is a LIN gate immediately fed by a MUL gate, and λ is not the output gate;

4. λ is a LIN gate immediately fed by a variable, and λ is the output gate; and

5. λ is a LIN gate immediately fed by a MUL gate, and λ is the output gate.

When λ is not the output gate (i.e., cases 1, 2 and 3), we show inductively how to find an approximation $\hat{h}$ of $h = g/\bar{g}$ so that $\mathbf{E}\left[\left\|h - \hat{h}\right\|\right] \leq s\tau/\bar{g}$ where s is the number of gates in g, and

$$\tau = \epsilon/12n^2.$$

We will find it useful in case 1 to also prove as part of our inductive hypothesis that $\mathbf{E}\left[\left\|h - \hat{h}\right\|\right] \leq 5$. When λ is the output gate (cases 4 and 5) we show how to find an approximation $\hat{f}$ of $g = f$, so that $\mathbf{E}\left[\left\|f - \hat{f}\right\|\right] \leq \epsilon/4$.

Case 1

In this case, λ is a MUL gate. Thus, we can express g as a product of its inputs, $g = g_1g_2$. We assume inductively that, for $i = 1, 2$, approximations $\hat{h}_i$ are available for $h_i = g_i/\bar{g}_i$ with the property that

$$\mathbf{E}\left[\left\|h_i - \hat{h}_i\right\|\right] \leq s_i\tau/\bar{g}_i.$$

Here, s_i is the number of gates in g_i, so that $s = s_1 + s_2 + 1$ is the number of gates in g.

We use $\hat{h}_1$ and $\hat{h}_2$ to approximate $h = g/\bar{g}$. Clearly, by independence, $h = g/\bar{g} = g_1g_2/\bar{g}_1\bar{g}_2 = h_1h_2$, so we let $\hat{h} = \hat{h}_1\hat{h}_2$ in this case.

Then the average error of $\hat{h}$ can be computed as follows:

$$
\begin{aligned}
\mathbf{E}\left[\left\|h - \hat{h}\right\|\right] &= \mathbf{E}\left[\left\|h_1h_2 - \hat{h}_1\hat{h}_2\right\|\right] \\
&= \mathbf{E}\left[\left\|h_1(h_2 - \hat{h}_2) + h_2(h_1 - \hat{h}_1) - (h_1 - \hat{h}_1)(h_2 - \hat{h}_2)\right\|\right] \\
&\leq \mathbf{E}\left[|h_1|\left\|h_2 - \hat{h}_2\right\| + |h_2|\left\|h_1 - \hat{h}_1\right\| + \left\|h_1 - \hat{h}_1\right\|\left\|h_2 - \hat{h}_2\right\|\right].
\end{aligned}
$$

Note that $\bar{h}_1 = \bar{h}_2 = 1$, and that h_1 and h_2 are nonnegative functions. Thus, by independence, and by our inductive assumptions about the accuracy of $\hat{h}_1$ and $\hat{h}_2$, it follows that $\mathbf{E}\left[\left\|h - \hat{h}\right\|\right]$ is bounded by

$$\frac{s_1\tau}{\bar{g}_1} + \frac{s_2\tau}{\bar{g}_2} + \frac{s_1s_2\tau^2}{\bar{g}_1\bar{g}_2}. \tag{7.6}$$

Since $\bar{g} = \bar{g}_1\bar{g}_2 \leq \bar{g}_i$ for $i = 1, 2$, this is at most

$$\frac{s_1\tau + s_2\tau + s_1s_2\tau^2}{\bar{g}} \leq \frac{s_1\tau + s_2\tau + \tau}{\bar{g}}$$
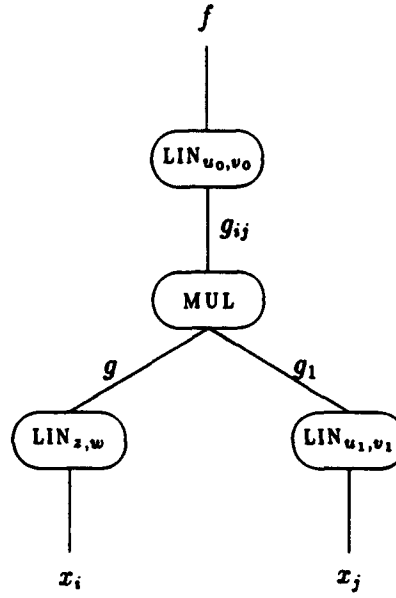
Figure 6: The decomposition of f used in Case 2.

since each $s_i \leq 3n$, and since $\tau \leq 1/9n^2$. This gives the desired bound of $s\tau/\bar{g}$ for Case 1.

Alternatively, note that $\bar{g}_1 \geq \epsilon/5n$ since g_1 is subsumed by an uncle of some influential variable x_j relevant to g_2, and by Lemma 7.1. Thus, since $s_1 \leq 3n$, $s_1\tau/\bar{g}_1 \leq 5/4$ by our choice of τ. Similarly, $s_2\tau/\bar{g}_2 \leq 5/4$, and therefore, the bound given in equation (7.6) implies that $\mathbf{E}\left[\left\|h - \hat{h}\right\|\right] \leq 65/16 < 5$.

Case 2

In this case, λ is a LIN$_{zw}$ gate immediately fed by some variable x_i so that $g = z + wx_i$. Also, λ is an input to some MUL gate Γ_{ij} for some index j. Thus, $g_{ij} = gg_1$ for some subformula g_1 to which x_j is relevant. We can decompose g_1 in terms of x_j, and write $g_1 = u_1 + v_1x_j$. Similarly, $f = u_0 + v_0g_{ij}$ for f's decomposition in terms of g_{ij}. (This decomposition of f is shown in Figure 6.)

Thus, $f = u_0 + v_0gg_1 = u_0 + v_0(z + wx_i)(u_1 + v_1x_j)$, so

$$B_j = v_0v_1(z + wx_i)$$

$$A_{ij} = v_0v_1w.$$

Also, $(B_j|x_i \leftarrow 0) = v_0v_1z$. Let $\alpha = \mathbf{E}[B_j|x_i \leftarrow 0]$, $\beta = \bar{A}_{ij}$ and $\gamma = \bar{B}_j$. Then by independence, $\alpha = \bar{v}_0\bar{v}_1z$, $\beta = \bar{v}_0\bar{v}_1w$, and $\gamma = \bar{v}_0\bar{v}_1(z + wp_i)$. (Recall that p_i is the probability that bit x_i is 1.) Let $\hat{\alpha}$, $\hat{\beta}$ and $\hat{\gamma}$ be respective estimates of these quantities. (The quantity α can be estimated using the fact that $\alpha = \mathbf{E}[(f|x_i \leftarrow 0, x_j \leftarrow 1) - (f|x_i \leftarrow 0, x_j \leftarrow 0)]$. Thus, $|\alpha - \hat{\alpha}| \leq a\epsilon^4/n^5$.)

Then it can be seen that

$$\frac{\alpha + \beta x_i}{\gamma} = \frac{z + w x_i}{z + w p_i} = \frac{g}{\bar{g}} = h,$$

so let $\hat{h} = (\hat{\alpha} + \hat{\beta} x_i)/\hat{\gamma}$ in this case.

The next fact will be useful for analyzing the error of $\hat{h}$.

Lemma 7.13 *For b and $\hat{b}$ nonzero,*

$$\left| \frac{\hat{a}}{\hat{b}} - \frac{a}{b} \right| \le \frac{|\hat{a} - a|}{|\hat{b}|} + \left| \frac{a}{b} \right| \cdot \frac{|\hat{b} - b|}{|\hat{b}|}.$$

Proof:

$$\left| \frac{\hat{a}}{\hat{b}} - \frac{a}{b} \right| = \left| \frac{\hat{a}}{\hat{b}} - \frac{a}{b} + \frac{a}{b} - \frac{a}{b} \right| \le \frac{|\hat{a} - a|}{|\hat{b}|} + \left| \frac{ab - a\hat{b}}{b\hat{b}} \right| = \frac{|\hat{a} - a|}{|\hat{b}|} + \left| \frac{a}{b} \right| \cdot \frac{|\hat{b} - b|}{|\hat{b}|}.$$

∎

We are now ready to analyze the error of $\hat{h}$. First, since x_j is influential, $|\hat{\gamma}| \ge (\epsilon/5n)$. By Lemma 7.13,

$$\mathbf{E}\left[\left| \hat{h} - h \right|\right] \le \mathbf{E}\left[\frac{\left| \hat{\alpha} - \alpha + (\hat{\beta} - \beta) x_i \right|}{|\hat{\gamma}|} + |h| \cdot \frac{|\hat{\gamma} - \gamma|}{|\hat{\gamma}|} \right].$$

Clearly, h is nonnegative and $\bar{h} = 1$. Thus,

$$
\begin{aligned}
\mathbf{E}\left[\left| \hat{h} - h \right|\right] &\le \frac{5n}{\epsilon} \cdot \left(|\hat{\alpha} - \alpha| + \left| \hat{\beta} - \beta \right| + |\hat{\gamma} - \gamma| \right) \\
&\le \frac{5n}{\epsilon} \cdot \frac{3a\epsilon^4}{n^5} \\
&= \frac{15a\epsilon^3}{n^4} \le \tau \le \frac{\tau}{\bar{g}}
\end{aligned}
$$

since $\bar{g} \le 1$. Since g contains one gate, this satisfies our inductive hypothesis in case 2.

Case 3

In this case, λ is a LIN$_{zw}$ gate which has as input some MUL gate Γ_{ij} so that $g = z + w g_{ij}$. Then g_{ij} can be expressed as a product of its inputs $g_{ij} = g_1 g_2$, where x_i and x_j are relevant to g_1 and g_2, respectively. Decomposing g_1 and g_2, we can write $g_1 = u_1 + v_1 x_i$ and $g_2 = u_2 + v_2 x_j$. Gate λ immediately feeds some MUL gate Γ_{ik}. We can write g_{ik} as a product of its inputs $g_{ik} = g g_3$, where g_3 is some subformula which contains x_k. Decomposing g_3, we can write $g_3 = u_3 + v_3 x_k$. Finally, we can decompose f in terms of g_{ij} so that $f = u_0 + v_0 g_{ij}$. (All this is summarized in Figure 7.)
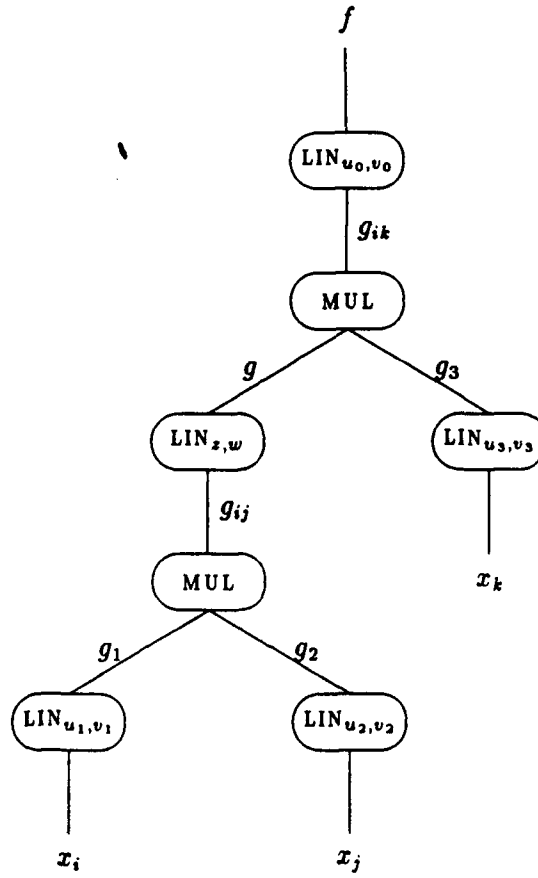
Figure 7: The decomposition of f used in Case 3.

Thus, we have that

$$B_i = v_0 v_1 w g_2 g_3$$

$$B_k = v_0(z + w g_1 g_2) v_3$$

$$A_{ij} = v_0 v_1 w v_2 g_3$$

$$A_{jk} = v_0 v_3 w g_1 v_2.$$

Let $\alpha = \bar{B}_i \cdot \bar{A}_{jk}$, and let $\beta = \bar{B}_k \cdot \bar{A}_{ij}$. Then, by independence, and since $g_{ij} = g_1 g_2$,

$$\alpha = \bar{v}_0^2 \bar{v}_1 \bar{v}_2 \bar{v}_3 w \bar{g}_3 (w \bar{g}_{ij})$$

$$\beta = \bar{v}_0^2 \bar{v}_1 \bar{v}_2 \bar{v}_3 w \bar{g}_3 (z + w \bar{g}_{ij}).$$

Let $\hat{\alpha} = \hat{B}_i \cdot \hat{A}_{jk}$ and $\hat{\beta} = \hat{B}_k \cdot \hat{A}_{ij}$.

We assume inductively that an approximation $\hat{h}_0$ is available for $h_0 = g_{ij}/\bar{g}_{ij}$. We assume

that $\mathbf{E}\left[\left|h_0 - \hat{h}_0\right|\right] \le \min(5, (s-1)\tau/\bar{g}_{ij})$, where s is the number of gates occurring in g.

Let $\gamma = \alpha/\beta$, and let $\hat{\gamma} = \hat{\alpha}/\hat{\beta}$. Then $\gamma = w\bar{g}_{ij}/(z + w\bar{g}_{ij}) = w\bar{g}_{ij}/\bar{g}$. Thus,

$$h = g/\bar{g} = 1 - \gamma + \gamma h_0.$$

Naturally, then, we let

$$\hat{h} = 1 - \hat{\gamma} + \hat{\gamma}\hat{h}_0$$

in this case. We show that $\hat{h}$ has the desired accuracy.

First,

$$\left|\hat{\alpha} - \alpha\right| \le \left|\hat{B}_i\right|\left|\hat{A}_{jk} - \bar{A}_{jk}\right| + \left|\bar{A}_{jk}\right|\left|\hat{B}_i - \bar{B}_i\right| \le 2a\epsilon^4/n^5.$$

Similarly, $\left|\hat{\beta} - \beta\right| \le 2a\epsilon^4/n^5$.

Note that $|\gamma| = |w| \cdot \bar{g}_{ij}/\bar{g} \le 1/\bar{g}$. Also, since all the variables are influential, $|\beta| \ge (\epsilon/5n)^3$ (since $|\bar{A}_{ij}| \ge |\bar{B}_i||\bar{B}_j|$); thus,

$$\left|\hat{\beta}\right| \ge (\epsilon/5n)^3 - 2a\epsilon^4/n^5 \ge (5^{-3} - 2a)(\epsilon/n)^3.$$

So, by Lemma 7.13, we have

$$
\begin{aligned}
|\hat{\gamma} - \gamma| &\le \frac{1}{\left|\hat{\beta}\right|} \cdot \left(|\hat{\alpha} - \alpha| + |\gamma|\left|\hat{\beta} - \beta\right|\right) \\
&\le \frac{1}{5^{-3} - 2a} \cdot \left(\frac{n}{\epsilon}\right)^3 \cdot \left(\frac{|\hat{\alpha} - \alpha| + \left|\hat{\beta} - \beta\right|}{\bar{g}}\right) \\
&\le \frac{1}{\bar{g}} \cdot \left(\frac{4a}{5^{-3} - 2a} \cdot \frac{\epsilon}{n^2}\right) \\
&\le \frac{\tau}{7\bar{g}}.
\end{aligned}
$$

So the error of $\hat{h}$ can be computed as follows:

$$
\begin{aligned}
\mathbf{E}\left[\left|h - \hat{h}\right|\right] &= \mathbf{E}\left[\left|\hat{\gamma} - \gamma + \gamma h_0 - \hat{\gamma}\hat{h}_0\right|\right] \\
&\le |\gamma - \hat{\gamma}| + |\gamma - \hat{\gamma}| \cdot \mathbf{E}[|h_0|] + |\gamma| \cdot \mathbf{E}\left[\left|h_0 - \hat{h}_0\right|\right] + |\gamma - \hat{\gamma}| \cdot \mathbf{E}\left[\left|h_0 - \hat{h}_0\right|\right].
\end{aligned}
$$

Since $\mathbf{E}[|h_0|] = 1$, and by our inductive assumption on the error of $\hat{h}_0$, this is at most

$$
\begin{aligned}
7|\gamma - \hat{\gamma}| + |\gamma| \cdot \mathbf{E}\left[\left|h_0 - \hat{h}_0\right|\right] &\le 7|\gamma - \hat{\gamma}| + \frac{|w|\bar{g}_{ij}}{\bar{g}} \cdot \frac{(s-1)\tau}{\bar{g}_{ij}} \\
&\le \frac{\tau + (s-1)\tau}{\bar{g}} = s\tau/\bar{g}
\end{aligned}
$$
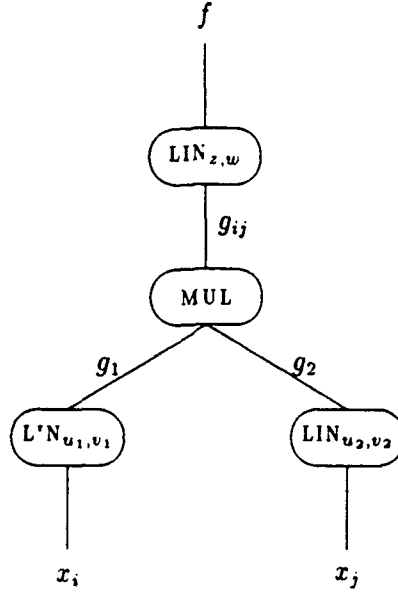
Figure 8: The decomposition of f used in Case 5.

as desired.

Case 4

In this very easy case, $f = \text{LIN}_{zw}(x_1)$ for some z, w. Since $f = z + wx_1$, $\bar{B}_1 = B_1 = w$. Let $\hat{w} = \hat{B}_1$, and let $\hat{z}$ be an estimate of $z = \mathbf{E}[f|x_1 \leftarrow 0]$. (Recall that an estimate of this latter quantity was made in calculating $\bar{B}_1$. Thus, $|z - \hat{z}| \leq a\epsilon^4/n^5$.) Finally, let $\hat{f} = \hat{z} + \hat{w}x_1$. Then

$$\mathbf{E}\left[\left|f - \hat{f}\right|\right] \leq |z - \hat{z}| + |w - \hat{w}| \leq 2a\epsilon^4/n^5 \leq \epsilon/4$$

as desired.

Case 5

In this case, λ is a LIN_{zw} gate computing the formula's output f. Gate λ receives its input from some MUL gate Γ_{ij}. Thus, $f = z + wg_{ij}$, and g_{ij} computes the product $g_{ij} = g_1 g_2$, where variables x_i and x_j are relevant to g_1 and g_2, respectively. (Refer to Figure 8.)

We assume inductively that an approximation $\hat{h}_0$ has been computed for $h_0 = g_{ij}/\bar{g}_{ij}$, and that the accuracy of $\hat{h}_0$ is such that $\mathbf{E}\left[\left|\hat{h}_0 - h_0\right|\right] \leq \min(5, (s-1)\tau/\bar{g}_{ij})$ where s is the number of gates occurring in f. We wish to use $\hat{h}_0$ to compute an approximation $\hat{f}$ so that $\mathbf{E}\left[\left|\hat{f} - f\right|\right] \leq \epsilon/4$.

We can decompose each subformula g_i so that $g_1 = u_1 + v_1 x_i$ and $g_2 = u_2 + v_2 x_j$. Then we

have:

$$B_i = wv_1g_2$$

$$B_j = wv_2g_1$$

$$A_{ij} = wv_1v_2.$$

Let $\alpha = \bar{f}$, and let $\beta = \bar{B}_i \cdot \bar{B}_j / \bar{A}_{ij}$. Let $\hat{\beta} = \hat{B}_i \cdot \hat{B}_j / \hat{A}_{ij}$, and let $\hat{\alpha}$ be an estimate of $\bar{f}$ so that $|\alpha - \hat{\alpha}| \leq \epsilon/24n^2$ with probability at least $1 - \delta/4$. Such an estimate is easily computed since $\bar{f}$ is just the probability that a positive example is received from the examples oracle. (Actually, we can only sample according to the "true" target formula, rather than the one derived in Stage II, and here regarded as the target. The additional error introduced by this fact has been folded into the stated accuracy, as was done for our estimates $\hat{B}_i$, etc.)

Note that $\alpha = z + w\bar{g}_{ij}$, and $\beta = w\bar{g}_1\bar{g}_2 = w\bar{g}_{ij}$. Thus, it is easily seen that

$$f = (\alpha - \beta) + \beta h_0,$$

and we let

$$\hat{f} = (\hat{\alpha} - \hat{\beta}) + \hat{\beta}\hat{h}_0.$$

We show that $\hat{f}$ has the desired accuracy.

First, note that

$$\left|\hat{B}_i\hat{B}_j - \bar{B}_i\bar{B}_j\right| \leq \left|\hat{B}_i\right|\left|\hat{B}_j - \bar{B}_j\right| + \left|\bar{B}_j\right|\left|\hat{B}_i - \bar{B}_i\right|$$
$$\leq 2a\epsilon^4/n^5.$$

Clearly, $|\beta| \leq 1$, so $|\bar{A}_{ij}| \geq |\bar{B}_i||\bar{B}_j| \geq (\epsilon/5n)^2$. Thus, $\left|\hat{A}_{ij}\right| \geq (\epsilon/5n)^2 - a\epsilon^4/n^5 \geq (5^{-2} - a)(\epsilon/n)^2$. Thus, by Lemma 7.13,

$$\left|\hat{\beta} - \beta\right| \leq \frac{1}{\left|\hat{A}_{ij}\right|} \cdot \left(\left|\hat{B}_i\hat{B}_j - \bar{B}_i\bar{B}_j\right| + |\beta|\left|\hat{A}_{ij} - \bar{A}_{ij}\right|\right)$$
$$\leq \frac{3a}{5^{-2} - a} \cdot \frac{\epsilon^2}{n^3}.$$

Since $|\beta| \leq 1$, we assume without loss of generality that $\left|\hat{\beta}\right| \leq 1$. Thus, the average error of $\hat{f}$ can then be computed as follows:

$$\mathbf{E}\left[\left|f - \hat{f}\right|\right] = \mathbf{E}\left[\left|\alpha - \hat{\alpha} + \hat{\beta} - \beta + \beta h_0 - \hat{\beta}\hat{h}_0\right|\right]$$
$$\leq \mathbf{E}\left[\left|\alpha - \hat{\alpha}\right| + \left|\beta - \hat{\beta}\right| + \left|\beta - \hat{\beta}\right||h_0| + |\beta|\left|h_0 - \hat{h}_0\right| + \left|\beta - \hat{\beta}\right|\left|h_0 - \hat{h}_0\right|\right]$$

$$\leq \quad |\alpha - \hat{\alpha}| + 7\left|\beta - \hat{\beta}\right| + |\beta| \cdot \mathbf{E}\left[\left|h_0 - \hat{h}_0\right|\right]$$

$$\leq \quad \frac{\epsilon}{24n^2} + \frac{21a\epsilon^2}{(5^{-2} - a)n^3} + w(s-1)\tau$$

$$\leq \quad s\tau \leq 3n\tau \leq \epsilon/4$$

as desired. Here we have used the fact that $\mathbf{E}\left[|h_0|\right] = 1$, and our inductive bound on the error of $\hat{h}_0$.

Combined with the previous cases, this completes the induction, and proves that the final computed hypothesis $\hat{f}$ has error at most $\mathbf{E}\left[\left|f - \hat{f}\right|\right] \leq \epsilon/4$.

3-7.6 Putting it all together

Thus, we showed in Stage I how to eliminate sticky and uninfluential variables from the target formula f, yielding another formula f_I with $\mathbf{E}\left[|f - f_I|\right] \leq 2\epsilon/3$.

In Stage II, we regarded f_I as the target, and showed how to find an approximate skeleton σ of f_I; in particular, we showed that there exist z, w values for σ which result in a formula f_{II}. This formula is very close to f_I; we showed that $|f_I - f_{II}|$ is much smaller than $\epsilon/12$ on all inputs, and so certainly $\mathbf{E}\left[|f_I - f_{II}|\right] \leq \epsilon/12$.

Finally, in Stage III, we regarded f_{II} as the target, causing a slight degradation in the accuracy of our estimates. Nevertheless, we described a technique for approximating the z, w values for f_{II}, giving a final formula $\hat{f}$ with $\mathbf{E}\left[\left|\hat{f} - f_{II}\right|\right] \leq \epsilon/4$. It follows immediately that $\mathbf{E}\left[\left|f - \hat{f}\right|\right] \leq \epsilon$ as desired. In other words, $\hat{f}$ is an ϵ-good model of probability for f.

All of the operations described are clearly polynomial time, and the sample size is also polynomial. The sample was needed to estimate the probability that each bit is 1, and the expected value of f when zero, one or two unsticky variables are hardwired to fixed values. For the accuracy needed, we can use Chernoff bounds (Lemma 2-3.6) to show that a sample of size $O((n^{22}/\epsilon^{18}) \cdot \log(n/\delta))$ suffices.

After the sample is drawn our algorithm records the obtained estimates of these values. All of the remaining operations of the algorithm take negligible time compared to the time needed to compute and record these values from the (unfortunately quite large) sample.

Thus we have:

Theorem 7.14 *There exists an algorithm with the following properties: Given $\epsilon, \delta > 0$ and access to random examples chosen according to a product distribution and classified randomly according to some read-once real formula f, the algorithm outputs an ϵ-good model of probability for f with probability at least $1 - \delta$. The sample size needed by the algorithm is $O((n^{22}/\epsilon^{18}) \cdot \log(n/\delta))$, and its running time is $O((n^{24}/\epsilon^{18}) \cdot \log(n/\delta))$.*

From the comments made at the beginning of this section, we have as corollary:

Corollary 7.15 *There exists a polynomial-time algorithm that PAC-learns the class of read-once Boolean formulas against any product distribution.*

3-8   Conclusion and open problems

In this chapter, we have described polynomial-time algorithms for learning various classes of read-once formulas in a number of settings. Our algorithms are based on a simple statistical method of observing the formula's behavior under various perturbations of the target distribution.

The main open question is to determine how far this apparently powerful method can be extended. In particular, can it be applied to formulas which are not read-once? Can it be extended beyond product distributions? Are there other classes entirely different to which it might be extended, such as decision trees, or finite automata?

Considering the classes described in this chapter, can the algorithms described be improved? (It seems that this should certainly be the case for the algorithm of Section 3-7.) Turning the question around, can we find good, non-trivial lower bounds for these problems? It is unclear what such a lower-bound proof would look like, especially since, in the PAC model, much smaller sample sizes are known to suffice in a computationally unbounded setting. (This follows, for instance, from Occam's Razor of Blumer et al. [13].)

Finally, can our algorithms be extended to the so-called two-oracle model? In the one-oracle model (considered exclusively in this chapter), the learner receives both positive and negative examples from a random source of examples, and must perform well as measured against this single distribution. In the two-oracle model, the learner has two random sources of examples, one that provides just positive examples, and the other providing just negative examples. The learner must perform well against both distributions (i.e., both the distribution on the positive, and the distribution on the negative examples). In the distribution-free model, Haussler et al. [38] show that these two models are equivalent. However, their proof falls apart when the form of the target distribution is restricted. Kearns et al. [51] and Pagallo and Haussler [66] have shown that read-once formulas in disjunctive normal form are efficiently learnable against the uniform distribution in the two-oracle model. Can the results in this chapter be similarly extended?

Efficient Distribution-free Learning
of Probabilistic Concepts

4-1 Introduction

Consider the following scenarios:

A meteorologist is attempting to predict tomorrow's weather as accurately as possible. He measures a small number of presumably relevant parameters, such as the current temperature, barometric pressure, and wind speed and direction. He then makes a forecast of the form "chances for rain tomorrow are 70%." The next day it either rains or it does not rain.

A statistician wishes to compile an approximate rule for predicting when students will be admitted to a particular college. There are some students whose record is so strong they will be accepted regardless of which admissions officer reviews their file; similarly, there are others who are categorically rejected. For many students, however, their admission may be highly dependent on the particular admissions officer that evaluates their application; thus the best model for the chances of these borderline students involves a probability of acceptance. However, every student is either accepted or rejected.

A physicist is attempting to determine the orientation of spin for particles in a certain magnetic field. Presumably, the orientation of spin is at least partially determined by a genuinely random process of Nature. The spin of any particle is always oriented either up or down.

We wish to produce a good model for the recognition of common objects such as chairs. For most objects in the world, there is nearly universal agreement as to whether that object

is a chair or a non-chair. There do exist, however, a few objects that provoke widespread disagreement, such as stools and benches. This is due to the fact that the concept of "chair" is not absolute, and philosophical boundaries of this concept may be exposed by both naturally occurring and artificially constructed objects. Most young children, however, are not explicitly told about such definitional shortcomings; they are simply told whether or not something is a chair.

There are some obvious common themes in each of the above situations. First, in each there is *uncertain* or probabilistic behavior. This uncertainty may arise for radically different philosophical reasons. For example, in the case of the meteorologist, it could be that while the weather is in principle a deterministic process, the parameters measured by the meteorologist and the limited accuracy of these measurements are insufficient to determine this process. In the case of the physicist, the electron spin is believed to be governed to some degree by a truly random process. In the case of the statistician, the uncertainty arises from the diversity of human behavior, and in the case of chair recognition, a probability may model the philosophical difficulties of providing a deterministic definition to an inherently uncertain or "fuzzy" concept.

A second theme that is common to each of these settings is the fact that even though the best model may be a conditional probability $p(x)$ that the event (rain, acceptance to the college, etc.) occurs given x (where x represents the measured weather variables or a student's application), the observer only witnesses whether or not the event occurs. Thus, examples are of the form $(x, 0)$ or $(x, 1)$ — *not* $(x, p(x))$ — and the $\{0, 1\}$ label provided with x is distributed according to the conditional probability $p(x)$. Furthermore, we should not expect to be able to compute even an *estimate* of $p(x)$ from the given $\{0, 1\}$-labeled examples, since in general we are unlikely to ever see the same x twice (each day's weather is at least slightly different, as is each student's application).

Finally, although there is uncertainty in each of these settings, there is also some *structure* to this uncertainty. For instance, days with nearly identical atmospheric conditions and students with very similar high school records can be expected to have nearly equal probabilities of rain and acceptance to the college, respectively. We also expect some inputs x to be assigned conditional probabilities that are very near 0 or 1; for example, days on which the sky is cloudless or students with straight A's. This structured behavior strongly distinguishes these learning scenarios from a "noisy" setting, such as that considered by Angluin and Laird [9], Kearns and Li [50], and Sloan [81]. In a model of learning with noise, the noise is typically "white" (that is, all inputs have either an equal probability of corruption or a probability determined by an adversary), and the noise is regarded as something an algorithm wishes to "filter out" in an attempt to uncover some underlying *deterministic* concept. In the examples given above, the probabilistic behavior is both *structured* (possibly in a manner that can be exploited by a

learning algorithm) and *inherently part of the underlying phenomenon*. Thus, whenever possible we do not wish to filter this probabilistic behavior out of the hypothesis, but rather to *model* it.

In this chapter we wish to study a model of learning in such uncertain environments. We formalize these settings by introducing the notion of a *probabilistic concept* (or *p-concept*). A p-concept c over a domain set X is simply a mapping $c : X \rightarrow [0,1]$. For each $x \in X$, we interpret $c(x)$ as the probability that x is a positive example of the p-concept c. Following the discussion above, a learning algorithm in this framework is attempting to infer something about the underlying target p-concept c solely on the basis of labeled examples (x, b), where $b \in \{0, 1\}$ is a bit generated randomly according to the conditional probability $c(x)$, i.e., $b = 1$ with probability $c(x)$.

The value $c(x)$ may be viewed as a measure of the degree to which x exemplifies some concept c. In this sense, p-concepts are quite similar to the related notion of a *fuzzy set*, a kind of "set" whose boundaries are fuzzy or unclear, and whose formal definition is nearly identical to that of a p-concept. An axiomatic theory of fuzzy sets was introduced by Zadeh [89], and they have since received much treatment by researchers in the field of pattern recognition. See Kandel's book [47] for a good introduction.

We distinguish two possible goals for a learning algorithm in the p-concept model. The first and easier goal is that of *label prediction*: the algorithm wishes to output a hypothesis that maximizes the probability of correctly predicting the $\{0, 1\}$ label generated by c on an input x. We call this kind of learning *decision-rule learning*, since we are not primarily concerned with actually modeling the underlying uncertainty but instead wish to accurately predict the observable $\{0, 1\}$ outcome of this uncertainty. The more difficult and more interesting goal is that of finding a good *model of probability*. Here the algorithm wishes to output a hypothesis p-concept $h : X \rightarrow [0, 1]$ that is a good real-valued approximation to the target c; thus, we want $|c(x) - h(x)|$ to be small for most inputs x. Following the motivation given above, we are mainly concerned with this latter notion of learning.

To model the aforementioned structure of the target p-concept, we study the learnability of classes of p-concepts that obey natural mathematical properties intended to model some realistic environments. As a simple example, in constructing a p-concept model of the subjective notion of "tall," it is reasonable to assume that $x \geq y$ implies $c(x) \geq c(y)$ (where x represents height) — the taller a person actually is, the higher the percentage of people who will agree he is tall (or the greater the "degree of tallness" we wish to assign). This motivates us to consider learning the class C of all non-decreasing p-concepts over the positive real line. In general, we wish to study the learnability of p-concept classes that are restricted in such a way as to plausibly capture some realistic situation, but are not so restricted as to make the learning problem trivial

or uninteresting.

We adopt from the Valiant model for learning deterministic concepts [83] the emphasis on learning algorithms that are both efficient (in the sense of polynomial time) and general (in the sense of working for the largest possible p-concept classes and against any probability distribution over the domain). After formalizing the learning model and the two possible goals for a learning algorithm (decision-rule learning and model-of-probability learning), we embark on a systematic study of techniques for designing efficient algorithms for learning p-concepts and the underlying theory of the p-concept model.

We begin by giving examples of efficient algorithms producing a good model of probability that employ what we call the *direct approach*; the analyses of these algorithms give first-principles arguments that the output hypothesis is good. These include algorithms for arbitrary non-decreasing functions motivated above, a probabilistic analog of Rivest's decision lists [72], and a class of "hidden-variable" p-concepts, motivated by settings such as weather prediction where the apparently probabilistic behavior may in part be due to the fact that some relevant quantities remain undiscovered.

We then consider the problem of *hypothesis testing* in the p-concept model. Working within the framework suggested by Haussler [36], we define a *loss function* that assigns a measure of goodness to any hypothesis p-concept on a $\{0, 1\}$-labeled sample. After proving that the *quadratic loss* measure is most appropriate for our setting, we then give an example of an efficient algorithm for finding a model of probability that first does some direct computation in order to narrow the search and then uses quadratic loss to choose the best hypothesis from among a small remaining pool. This algorithm learns a class of p-concepts in which only a small number of variables are relevant, but the dependence on these variables may be arbitrary.

Next we consider the related but more difficult issue of *uniform convergence* of a p-concept class. More precisely, how many $\{0, 1\}$-labeled examples must be taken before we have high confidence that every p-concept in the class has an empirical quadratic loss that accurately reflects its true performance as a model of probability? In a more general formulation, this question has received extensive consideration in the statistical pattern recognition literature, and its importance to learning has been demonstrated by many recent papers. We show that the sufficient sample size for uniform convergence is bounded above by the *quadratic loss dimension* of the p-concept class, a combinatorial measure derived from the *combinatorial dimension* discussed by Haussler [36] and other authors.

We then give efficient algorithms that apply the uniform convergence method (that is, take a large enough sample as dictated by the quadratic loss dimension, and find the hypothesis minimizing the empirical loss over the sample) in order to find a good model of probability. In particular, we prove the effectiveness of an algorithm for learning p-concepts represented by

linear combinations of d given basis functions. We then show that the quadratic loss dimension, when finite, is also a lower bound on the required sample size for learning any p-concept class with a model of probability; thus the quadratic loss dimension, when finite, characterizes the sample complexity of p-concept learning with a model of probability in the same way that the Vapnik-Chervonenkis (VC) dimension characterizes sample complexity in Valiant's model. (See Blumer et al.'s paper [14] for a full discussion of the VC-dimension.) However, we show that p-concept classes of *infinite* quadratic loss dimension may sometimes be learned efficiently, in contrast to classes of infinite VC-dimension in the Valiant model, which are not learnable in *any* amount of time. (Technically, this is not always true if "dynamic" sampling is allowed; see Linial, Mansour and Rivest's paper [58] for further details.)

We conclude with an investigation of Occam's Razor in the p-concept model. In the Valiant model, Blumer et al. [13] show that it suffices for learning to find a consistent hypothesis that is slightly shorter than the sample data. We look for analogies in our setting: namely, when does "data compression" imply a good model of probability? We formalize this question, and argue briefly that several of our algorithms can be interpreted as implementing a form of data compression.

The primary contribution of this research is that of providing positive results for efficient learnability in a natural and important extension to Valiant's model. This is significant because the Valiant model has been criticized for both its strong hardness results (and drought of powerful positive results), and for the unrealistic deterministic and noise-free view it takes of the concepts to be learned. While at first it may seem paradoxical that we are able to simultaneously generalize the model and obtain many positive results, this intuitively may be explained by the fact that since we generalize the form of the representations being learned, there are more ways in which concepts that capture some natural and realistic setting may be simply expressed. In contrast, since the Valiant model tends to emphasize concept classes based on standard circuit complexity, one is quickly led to study very powerful and apparently difficult classes such as disjunctive normal form Boolean expressions.

Another contribution of this research is in demonstrating the feasibility and practicality of the approach suggested by Haussler [36]. His work addressed the issue of sample complexity upper bounds in great generality, even encompassing the case where the input-output relation to be learned has no prescribed functional form. This generality prevents Haussler from obtaining either good sample size lower bounds or efficient learning algorithms; indeed, he cites both of these as important areas for further research. Our results may be regarded as a first demonstration of applying some of Haussler's general principles to a specific and realistic model in which computation time is of foremost significance.

4-2 The learning model

Let X be a set called the *domain* (or *instance space*). A *probabilistic concept* (or *p-concept*) is a real-valued function $c : X \to [0,1]$. When learning the p-concept c, the value $c(x)$ is interpreted as the *probability* that x exemplifies the concept being learned (i.e., the probability that x is a positive example). A *p-concept class* C is a family of p-concepts. On any execution, a learning algorithm for C is attempting to learn a distinguished *target* p-concept $c \in C$ with respect to a fixed but unknown and arbitrary *target distribution* D over X. We think of D as modeling the natural distribution of objects in the domain, and c represents the probabilistic concept to be learned in this domain.

(More formally, D is a probability measure on a σ-algebra of measurable subsets of X. We assume implicitly that all of the p-concepts considered are measurable functions with respect to this σ-algrebra on X, and the Borel σ-algebra on $[0,1]$.)

The learning algorithm is given access to an oracle EX that behaves as follows: EX first draws a point $x \in X$ randomly according to the distribution D. Then with probability $c(x)$, EX returns the labeled example $(x,1)$ and with probability $1 - c(x)$ it returns $(x,0)$. Thus, learning algorithms never have direct access to the conditional probabilities $c(x)$, but only to random examples whose labels are distributed according to these unknown probabilities.

Let h be a function mapping X into $\{0,1\}$; we call such a function a *decision rule*. We define the *predictive error* of h on c with respect to D, denoted $R_D(c,h)$, as the probability that h will misclassify a randomly drawn point from EX. If h minimizes $R_D(c,\cdot)$, then we say that h is a *best decision rule*, or a *Bayes optimal decision rule*, for c. We say that h is an ϵ-*good* decision rule for c if $R_D(c,h) \le R_D(c,\bar{h}) + \epsilon$, where $\bar{h}$ is a best decision rule. Thus we ask that h be nearly as good as the best decision rule for c.

The *projection* of the p-concept c is the function $\pi_c : X \to \{0,1\}$ that is 1 if $c(x) \ge 1/2$ and 0 if $c(x) < 1/2$. It is well known and easy to show that for any target p-concept c, its projection π_c is a Bayes optimal decision rule.

In this chapter we are primarily interested not in the problem of finding a good decision rule, but in that of producing an accurate real-valued approximation to the target p-concept itself. Thus, we wish to infer a good *model of probability* with respect to the target distribution. We say that a p-concept h is an (ϵ, γ)-*good model of probability* of c with respect to D if we have $\mathbf{Pr}_{x \in D}[|h(x) - c(x)| > \gamma] \le \epsilon$. Thus, the value of h must be near that of c on most points x.

We are now ready to describe our model for learning p-concepts. Let C be a p-concept class over domain X. We say that C is *learnable with a model of probability* (respectively, *learnable with a decision rule*) if there is an algorithm A such that for any target p-concept $c \in C$, for any target distribution D over X, for any inputs $\epsilon > 0$, $\delta > 0$ (and $\gamma > 0$ for learning with

a model of probability), algorithm A, given access to EX, halts and with probability at least $1 - \delta$ outputs a p-concept h that is an (ϵ, γ)-good model of probability (respectively, an ϵ-good decision rule) for c with respect to D. Note that this model of learning p-concepts generalizes Valiant's model for learning deterministic concepts.

We say that C is *polynomially learnable* if A runs in time polynomial in $1/\epsilon$, $1/\delta$ and, where appropriate, $1/\gamma$. Often the p-concept class C will be parameterized by a complexity parameter n, that is $C = \bigcup_{n \geq 1} C_n$, and all p-concepts in C_n share a common subdomain X_n. In such cases we also allow a polynomial dependence on n.

Our first lemma shows that a good model of probability can always be efficiently used as a good decision rule; thus, learning with a model of probability is a harder problem than decision-rule learning.

Lemma 2.1 *Let C be a class of p-concepts. If C is (polynomially) learnable with a model of probability, then C is (polynomially) learnable with a decision rule.*

Proof: To prove the lemma, we show that the projection of a good model of probability can be used as a good decision rule. In particular, we show that if h is an (ϵ, γ)-good model of probability, then π_h is an $(\epsilon + 2\gamma)$-good decision rule. Thus, by choosing ϵ and γ appropriately, an arbitrarily good decision rule can be found by the assumed algorithm for learning with a model of probability.

Let $x \in X$, and suppose $|h(x) - c(x)| \leq \gamma$. If $|c(x) - 1/2| > \gamma$, then clearly $\pi_h(x) = \pi_c(x)$. On the other hand, if $|c(x) - 1/2| \leq \gamma$, then it may be that $\pi_h(x) \neq \pi_c(x)$. However, the chance that $\pi_c(x)$ agrees with a random label for x (chosen according to c) is at most $1/2 + \gamma$, while the chance that $\pi_h(x)$ agrees with the random label is at least $1/2 - \gamma$.

Thus, the difference in predictive error between π_c and π_h (taken over a random choice of an instance and its label) is at most $\epsilon + (1 - \epsilon) \cdot 2\gamma \leq \epsilon + 2\gamma$. ∎

4-2.1 Alternative formulations

In addition to the formulation given above, there are various other natural ways of expressing the fact that some hypothesis p-concept h is "close" to the target c. For example, we might say that h is a good model of probability for c if the average difference between the two functions is small, i.e., if the quantity $\mathbf{E}_{x \in D}[|h(x) - c(x)|]$ is small. Alternatively, we might ask that the expected square of the difference between the functions be small.

As we will see in the following sections, these alternative definitions are sometimes easier to work with than the "official" definition given above. The next lemma shows that the three formulations are equivalent modulo polynomial-time computation.

Lemma 2.2 *Let h and c be p-concepts, and let D be a target distribution on domain X. Let $e_1 = \mathbf{E}_{x \in D}\left[|h(x) - c(x)|\right]$ and let $e_2 = \mathbf{E}_{x \in D}\left[(h(x) - c(x))^2\right]$. Then*

- *$e_2 \leq e_1 \leq \sqrt{e_2}$;*

- *for any $\gamma > 0$, h is both an $(e_1/\gamma, \gamma)$- and an $(e_2/\gamma^2, \gamma)$-good model of probability for c;*

- *if h is an (ϵ, γ)-good model of probability, then $e_1 \leq \epsilon + \gamma$, and $e_2 \leq \epsilon + \gamma^2$.*

Proof: Since $|h(x) - c(x)| \leq 1$ for all x, it is clear that $e_2 \leq e_1$. Also, by a convexity argument we have $(\mathbf{E}[Y])^2 \leq \mathbf{E}[Y^2]$ for any random variable Y. Thus, $e_1 \leq \sqrt{e_2}$.

Let $\gamma > 0$. Then by Markov's inequality,

$$\mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| > \gamma\right] \leq e_1/\gamma.$$

Similarly,

$$\mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| > \gamma\right] = \mathbf{Pr}_{x \in D}\left[(h(x) - c(x))^2 > \gamma^2\right] \leq e_2/\gamma^2.$$

These imply the second part of the lemma.

Finally, suppose h is an (ϵ, γ)-good model of probability. Then

$$e_1 \leq \mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| > \gamma\right] \cdot 1 + \mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| \leq \gamma\right] \cdot \gamma \leq \epsilon + (1 - \epsilon)\gamma \leq \epsilon + \gamma.$$

Similarly, $e_2 \leq \epsilon + \gamma^2$.   ∎

4-3   Efficient algorithms: The direct approach

In this section, we describe efficient algorithms for learning good models of probability based on first principles and proved correct by direct arguments. Later arguments will rely on an underlying theory of p-concept learning that is developed in subsequent sections. We begin with a p-concept class motivated by the problem of modeling "tallness" discussed in the introduction.

4-3.1   Increasing functions

Theorem 3.1 *The p-concept class of all nondecreasing functions $c : \mathbb{R} \to [0, 1]$ is polynomially learnable with a model of probability.*

Proof: We prove the result in slightly greater generality for any domain X linearly ordered by some ordering "$\leq$." Given positive ϵ, δ and γ, let $t = \lceil 4/\epsilon\gamma \rceil$, and let

$$s = \left\lceil \max\left\{ \frac{64 \ln(2^{21}/(\epsilon\gamma)^2\delta)}{\epsilon\gamma}, \frac{2\ln(4t/\delta)}{\gamma^2} \right\} \right\rceil.$$

Our algorithm begins by drawing a labeled sample of $m = st$ examples (x_i, b_i). The examples are sorted and reindexed so that $x_1 \leq \cdots \leq x_m$. In fact, we assume initially that no instance occurs twice in the sample so that $x_1 < \cdots < x_m$. Later, we show how this assumption can be removed.

The set X can naturally be partitioned into t disjoint intervals I_j, each containing exactly s instances of the sample; specifically, we let $I_1 = (-\infty, x_s]$; $I_j = (x_{(j-1)s}, x_{js}]$ for $j = 2, 3, \ldots, t-1$; and $I_t = (x_{(t-1)s}, \infty)$. For $1 \leq j \leq t$, let $\hat{p}_j = (1/s) \cdot \sum_{x_i \in I_j} b_i$. Thus, $\hat{p}_j$ is an estimate of the probability p_j that a random instance in I_j is labeled 1. Our algorithm outputs a step function h defined in a natural manner: for $x \in I_j$, we define $h(x) = \hat{p}_j$.

This algorithm clearly runs in polynomial time. We argue next that the output hypothesis h is an (ϵ, γ)-good model of probability (with high probability). Here are the high-level ideas: first, we show that (with high probability) each interval has weight approximately $\epsilon\gamma$ under the target distribution. Next we show that if c increases by roughly γ or less on the interval I_j, then h is close to c on all points in the interval. On the other hand, since c is nondecreasing and bounded between 0 and 1, c can increase by more than γ in at most $1/\gamma$ intervals; since these "bad" intervals have total weight at most ϵ, h is a good model of probability.

Specifically, we can apply the uniform convergence results of Vapnik and Chervonenkis [85] to show that each interval I_j has probability at most $\epsilon\gamma/2$. Let S be the set of all intervals on X. Then Theorem 2 of their paper shows that, with probability at least $1 - \delta/2$, for the sample size m chosen by our algorithm, the relative fraction of points of the sample occurring in *any* interval of S is within $\epsilon\gamma/4$ of the true weight of the interval under the target distribution. In particular, since each interval I_j contains $1/t \leq \epsilon\gamma/4$ of the instances in the sample, the weight of I_j under the target distribution is at most $\epsilon\gamma/2$.

(Technically, their results rely on certain measurability assumptions which depend on the choice of X. However, these assumptions are satisfied when $X = \mathbb{R}$.)

Let $q_j = c(x_{js})$ for $1 \leq j < t$, and let $q_0 = 0$ and $q_t = 1$. Then for $x \in I_j$, it is clear that $q_{j-1} \leq c(x) \leq q_j$ since c is nondecreasing. In particular, this is true for each $x_i \in I_j$. Thus, each point $x_i \in I_j$ is labeled 1 with probability $c(x_i) \geq q_{j-1}$, and so, for each j, $\hat{p}_j \geq q_{j-1} - \gamma/2$ with probability at least $1 - \delta/4t$; this follows from the fact that $s \geq (2/\gamma^2) \cdot \ln(4t/\delta)$, and by applying the additive form of Chernoff bounds given in Lemma 2-3.6. Similarly, $\hat{p}_j \leq q_j + \gamma/2$ with probability at least $1 - \delta/4t$. Thus, it follows that $q_{j-1} - \gamma/2 \leq \hat{p}_j \leq q_j + \gamma/2$ for all j with probability at least $1 - \delta/2$. Hence, if $q_j - q_{j-1} \leq \gamma/2$, then $|h(x) - c(x)| \leq \gamma$ for $x \in I_j$.

On the other hand, $q_j - q_{j-1}$ can exceed $\gamma/2$ for at most $2/\gamma$ values of j since c is nondecreasing, and bounded between 0 and 1. Since each of these "bad" intervals has probability weight at most $\epsilon\gamma/2$, the sum total probability of these intervals under D is at most ϵ. Thus, h is an (ϵ, γ)-good model of probability.

Finally, we show how to insure that the sample does not contain the same instance more than once. Such a situation could be problematic for our algorithm since it might cause some of the intervals defined above to be empty, or to contain too many sample points.

The idea is to replace the given domain X and target distribution D with a new domain X' and distribution D' under which the same instance is very unlikely to occur twice. In particular, we let $X' = X \times T$ and $D' = D \times U$ where U is the uniform distribution on the set $T = \{0, \ldots, 2^k - 1\}$, and $k = \lceil 2 \lg m + \lg(1/\delta) \rceil$. Then X' is linearly ordered under the lexicographic ordering (i.e., $(x, r) \leq (y, s)$ if and only if $x < y$, or $x = y$ and $r \leq s$). Also, the chance that any pair of instances are the same in a sample of size m drawn according to D' is at most $\binom{m}{2} \cdot 2^{-k} \leq m^2 \cdot 2^{-k-1} \leq \delta/2$.

In addition, given a random source of instances from X drawn according to D, we can easily simulate the random choice of instances from X' according to D': given $x \in X$, we simply draw a random number r uniformly from T, yielding an instance (x, r) with distribution D' (x's label is not altered). Thus, the previously described algorithm can be simulated (with δ replaced by $\delta/2$) on domain X'. If the same instance occurs twice in the sample, the algorithm simply fails — as argued above, this will happen with probability at most $\delta/2$. Thus, with probability at least $1 - \delta$, the algorithm returns an (ϵ, γ)-good hypothesis h (with respect to X'). This hypothesis can be used to estimate $c(x)$ for a given point $x \in X$ by randomly choosing $r \in T$ and evaluating $h((x, r))$. Although this yields a randomized hypothesis h', it remains true that the probability (over choices of $x \in X$ and the randomization of h') that h' differs by more than γ from c is at most ϵ. Thus, h' is an (ϵ, γ)-good model of probability if h is. ∎

This algorithm can be modified to learn with a model of probability any function over the real line with at most d extremal points; the running time is then polynomial in d, $1/\epsilon$, $1/\delta$ and $1/\gamma$.

In principle, the algorithm of Theorem 3.1 could be used to learn the p-concept class of non-decreasing functions with a decision rule (by applying Lemma 2.1). However, a much simpler and more efficient algorithm exists that we give in Section 4-5.

4-3.2   Probabilistic decision lists

We turn next to the problem of learning a probabilistic analog of Rivest's decision lists [72]. We define such lists with respect to a basis $\mathcal{F}_n$ of Boolean-valued functions on the domain $\{0, 1\}^n$. We assume always that $\mathcal{F}_n$ contains the constant function 1. Then a *probabilistic decision list* c *over basis* $\mathcal{F}_n$ is given by a list $(f_1, r_1), \ldots, (f_s, r_s)$, where each $f_i \in \mathcal{F}_n$, and each $r_i \in [0, 1]$. We also assume that f_s is the constant function 1. For any assignment x in the domain, $c(x)$ is defined to be r_j, where j is the least index for which $f_j(x) = 1$. In other words, the functions in $\mathcal{F}_n$ are tested one by one in the order specified by the list, until a function which evaluates

to 1 on x is encountered; the corresponding real number r_j is then the probability that x is labeled 1.

Rivest does not define decision lists with respect to a general basis as is done here. Rather, in his definition, a decision list only tests the values of monomials. That is, he defines decision lists specifically with respect to the basis consisting of all conjunctions of literals. He goes on to define the class k-DL of decision lists in which each monomial occurring in the list is a conjunction of k or fewer literals. Thus, this class is over the basis of all monomials of size at most k. Rivest describes an efficient algorithm for learning the class k-DL, when k is any fixed constant.

Below, we describe an efficient algorithm for learning a special class of probabilistic decision lists over any basis $\mathcal{F}_n$. The running time of this algorithm is polynomial in all of the usual parameters, in addition to $|\mathcal{F}_n|$, and the maximum time needed to evaluate any function f in $\mathcal{F}_n$. Thus, in particular, this implies a polynomial-time algorithm for the same basis considered by Rivest, namely, the set of all conjunctions of k or fewer literals, for k a fixed constant.

Let c be a probabilistic decision list over basis $\mathcal{F}_n$, given by the list $(f_1, r_1), \ldots, (f_s, r_s)$. For $\omega \in [0, 1]$, we say that c is a probabilistic decision list with ω-converging probabilities if $|r_i - \omega| \geq |r_{i+1} - \omega|$ for $1 \leq i < s$. Below, we describe an algorithm for inferring such lists when ω is known. As a special case, when $\omega = 0$, this algorithm can be used to learn probabilistic decision lists with decreasing probabilities, i.e., lists in which $r_i \geq r_j$ for $i \leq j$.

Perhaps the most natural case occurs when $\omega = 1/2$. In this case, we say that c is a probabilistic decision list with decreasing certainty since instances with the most certain outcomes (labels) are handled at the beginning of the list. For instance, a college's admissions process (see Section 4-1) might be naturally modeled in this manner as a list of criteria for determining admission, ordered by importance: for example, if the student has straight A's, then he should be admitted with 90% probability; otherwise, if he did poorly on his SAT's, then he should be rejected with 85% probability; otherwise, if he was class president, then he should be accepted with 75% probability; and so on. Note that the class of probabilistic decision lists with decreasing certainty includes the class of ordinary (deterministic) decision lists over the same basis.

We also note that the algorithm given below in Theorem 3.2 can be applied to learn ordinary decision lists when the supplied examples are "noisy." Specifically, consider the problem of learning a deterministic decision list c given by the list $(f_1, b_1), \ldots, (f_s, b_s)$ where each f_i is in the basis $\mathcal{F}_n$, and, since the list is deterministic, each $b_i \in \{0, 1\}$. Suppose further that the classification of each example is flipped randomly with probability $\eta < 1/2$. This random misclassification noise model is considered, for instance, by Angluin and Laird [9]. Note that the observed behavior in such a situation can be modeled naturally by the probabilistic decision list

Input: $\omega \in [0, 1]$

         basis $\mathcal{F}_n = \{f_1, \ldots, f_s\}$

         $\epsilon, \delta, \gamma > 0$

         access to random examples of a probabilistic decision list over basis $\mathcal{F}_n$

             with ω-converging probabilities

Output: with probability at least $1 - \delta$, an (ϵ, γ)-good model of probability

Procedure:

1   $L \leftarrow$ empty list
2   $J \leftarrow \{1, \ldots, s\}$
3   obtain a sample S of $m = \lceil (32s/\epsilon^3\gamma^2) \cdot \ln(2^{s+2}s/\delta) \rceil$ random examples
4   **repeat**
5   **if** $|\{(x, b) \in S : f_j(x) = 1\}| \leq m\epsilon/4s$ for some $j \in J$ **then**
6         $t \leftarrow j$
7         $\hat{p}_t \leftarrow 0$
8   **else**
9   **for** $j \in J$: $\hat{p}_j \leftarrow |\{(x, b) \in S : f_j(x) = 1 \wedge b = 1\}| \div |\{(x, b) \in S : f_j(x) = 1\}|$
10   choose t that maximizes $|\hat{p}_j - \omega|$
11     $L \leftarrow L, (f_t, \hat{p}_t)$
12     $S \leftarrow \{(x, b) \in S : f_t(x) = 0\}$
13     $J \leftarrow J - \{t\}$
14 **until** $J = \emptyset$
15 **output** L

Figure 1: An algorithm for learning probabilistic decision lists with ω-converging probabilities.

c' given by $(f_1, |b_1 - \eta|), \ldots, (f_s, |b_s - \eta|)$. That is, $c'(x)$ is the probability that x is labeled 1 by a noisy oracle for c. Clearly, c' is a probabilistic decision list with $1/2$-converging probabilities. Thus, we can apply the efficient learning algorithm for this class (described below) to obtain a good model of probability h for c'. If we choose $\gamma < 1/2 - \eta$, then it can be seen that the projection of h is a good approximation of c; that is, with probability at least $1 - \delta$, a hypothesis h is obtained for which $\mathbf{Pr}_{x \in D}[\pi_h(x) \neq c(x)] \leq \epsilon$. (Technically, this algorithm assumes that η, or an upper bound on η, is known. However, if no such bound is known, Angluin and Laird [9] give a technique for finding a good bound using a kind of "binary search.")

Thus, a corollary of Theorem 3.2 is a proof that deterministic decision lists are efficiently learnable even when the supplied examples are randomly misclassified with probability η. The running time is then polynomial in $1/(1 - 2\eta)$, in addition to the usual other parameters. This specifically answers an open question proposed by Rivest [72] concerning the learnability of decision lists in such a noisy setting.

Theorem 3.2 *Let $\omega \in [0, 1]$ be fixed, and let $\mathcal{F}_n$ be a basis of functions. Then the p-concept class of probabilistic decision lists over basis $\mathcal{F}_n$ with ω-converging probabilities is learnable with*

a model of probability (assuming both ω and $\mathcal{F}_n$ are known). Specifically, this class can be learned in time polynomial in $1/\epsilon$, $1/\gamma$, $1/\delta$, n, $|\mathcal{F}_n|$ and the maximum time needed to evaluate any function in $\mathcal{F}_n$.

Proof: Our learning algorithm for this p-concept class is shown in Figure 1. As usual, the algorithm begins by drawing a large sample S of size m which will be used to construct a hypothesis probabilistic decision list L. (Note that S and all subsets derived from S are *multisets* — they are "sets" which may contain multiple copies of the same example.)

Assume for convenience that the functions in $\mathcal{F}_n$ are indexed so that the target p-concept c is given by the list $(f_1, r_1), \ldots, (f_s, r_s)$. (Of course, the learning algorithm is not aware of this.) We also assume without loss of generality that every function in the basis $\mathcal{F}_n$ occurs in the target list so that $s = |\mathcal{F}_n|$.

Here is the intuition behind our algorithm: using the sample, we might estimate the probability p_i that a positive random example $(x, 1)$ is drawn, given that $f_i(x) = 1$. It can be shown to follow from the definition of ω-converging decision lists that $|p_1 - \omega| \geq |p_i - \omega|$ for all i. This suggests a technique for identifying the first variable in the list: if our estimates $\hat{p}_i$ are sufficiently accurate, we would expect $|\hat{p}_i - \omega|$ to be maximized when $i = 1$. This is the approach taken by our algorithm: the function f_i for which $|\hat{p}_i - \omega|$ is greatest is placed at the head of the hypothesis list. The remainder of the list is constructed iteratively using the part of the sample on which $f_i(x) = 0$.

For $I \subset \{1, \ldots, s\}$ and $j \in \{1, \ldots, s\}$, let $A(I, j)$ be the set of all instances x for which $f_j(x) = 1$ and $f_i(x) = 0$ for all $i \in I$. Let

$$u(I, j) = \mathbf{Pr}_{x \in D}\left[x \in A(I, j)\right]$$

and

$$v(I, j) = \mathbf{Pr}_{(x, b) \in EX}\left[b = 1 \mid x \in A(I, j)\right].$$

Also, let $\hat{u}(I, j)$ and $\hat{v}(I, j)$ be empirical estimates of these quantities derivable from the sample S in the obvious manner.

Let $I \subset \{1, \ldots, s\}$ and $j \in \{1, \ldots, s\}$ be fixed. Then, using the multiplicative form of Chernoff bounds given by Lemma 2-3.6, it follows that if $u(I, j) > \epsilon/2s$ then, since $m \geq (16s/\epsilon) \cdot \ln(2^{s+1}s/\delta)$,

$$\hat{u}(I, j) \geq \tfrac{1}{2} \cdot u(I, j)$$

with probability at least $1 - \delta/(s \cdot 2^{s+1})$. Furthermore, if $\hat{u}(I, j) > \epsilon/4s$, then the number of instances $x \in A(I, j)$ included in S is at least $m\epsilon/4s \geq (8/\epsilon^2\gamma^2) \cdot \ln(2^{s+2}s/\delta)$. Thus, applying

the additive form of Chernoff bounds, we see that

$$|v(I,j) - \hat{v}(I,j)| \le \epsilon\gamma/4$$

with probability at least $1 - \delta/2^{s+1}s$, assuming $\hat{u}(I,j) > \epsilon/4s$.

Thus, with probability at least $1 - \delta$, a sample S is chosen such that for all $I \subset \{1, \ldots, s\}$ and for all $j \in \{1, \ldots, s\}$, we have that

$$u(I,j) \le \max\left(\frac{\epsilon}{2s}, 2\hat{u}(I,j)\right), \tag{3.1}$$

and, whenever $\hat{u}(I,j) > \epsilon/4s$, we also have that

$$|v(I,j) - \hat{v}(I,j)| \le \frac{\epsilon\gamma}{4}. \tag{3.2}$$

We assume henceforth that all of the empirical estimates $\hat{u}(I,j)$ and $\hat{v}(I,j)$ satisfy the conditions described above. As just argued, this will be the case with probability at least $1 - \delta$. To complete the proof, we show that this assumption implies that the algorithm's output hypothesis h is an (ϵ, γ)-good model of probability.

Suppose h is given by the list $(f_{t_1}, r'_1), \ldots, (f_{t_s}, r'_s)$. Let $T_i = \{t_1, \ldots, t_i\}$. To prove that h is an (ϵ, γ)-good model of probability, we show that, for $1 \le i \le s$, either

$$\mathbf{Pr}_{x \in D}\left[x \in A(T_{i-1}, t_i)\right] \le \epsilon/2s \tag{3.3}$$

or

$$\mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| > \gamma \mid x \in A(T_{i-1}, t_i)\right] \le \epsilon/2. \tag{3.4}$$

Note that the sets $A(T_{i-1}, t_i)$ are disjoint. Thus, this implies

$$\begin{aligned}
\mathbf{Pr}_{x \in D}&\left[|h(x) - c(x)| > \gamma\right] \\
&= \sum_{i=1}^{s} \mathbf{Pr}_{x \in D}\left[|h(x) - c(x)| > \gamma \mid x \in A(T_{i-1}, t_i)\right] \cdot \mathbf{Pr}_{x \in D}\left[x \in A(T_{i-1}, t_i)\right] \\
&\le \epsilon
\end{aligned}$$

as can be seen by breaking the sum into two parts based on whether $\mathbf{Pr}_{x \in D}\left[x \in A(T_{i-1}, t_i)\right]$ exceeds or does not exceed $\epsilon/2s$.

Fix i, and consider the ith iteration of our algorithm. Prior to the extension of L at line 11, the hypothesis list is $(f_{t_1}, r'_1), \ldots, (f_{t_{i-1}}, r'_{i-1})$. Let $C_j = A(T_{i-1}, j)$. Also, let $p_j = v(T_{i-1}, j)$, and observe that, as defined in the figure, $\hat{p}_j = \hat{v}(T_{i-1}, j)$. This follows from the fact that, at this point in the execution of the algorithm, all examples (x, b) in S are such that $f_k(x) = 0$ for

$k \in T_{i-1}$.

Let t be as in the figure (i.e., $t = t_i$). If t was chosen at line 6, then $\hat{u}(T_{i-1}, t) \leq \epsilon/4s$, and so $u(T_{i-1}, t) \leq \epsilon/2s$ by equation (3.1). Thus, in this case, equation (3.3) holds by definition of $u(I, j)$.

Otherwise, for all $j \in J$, $\hat{u}(T_{i-1}, j) > \epsilon/4s$, and thus, $|p_j - \hat{p}_j| \leq \epsilon\gamma/4$ by equation (3.2). We wish to prove that equation (3.4) holds in this case, i.e., that

$$\mathbf{Pr}_{x \in D}\left[|\hat{p}_t - c(x)| > \gamma \mid x \in C_t\right] \leq \epsilon/2.$$

Let u be the smallest member of J. Then $p_u = r_u$ by definition of decision lists. Also, since c is given by a list with ω-converging probabilities, $|r_u - \omega| \geq |r_j - \omega|$ for $j \geq u$. Thus, by our choice of t, for $j \in J$,

$$|r_j - \omega| \leq |r_u - \omega| = |p_u - \omega| \leq |\hat{p}_u - \omega| + \epsilon\gamma/4 \leq |\hat{p}_t - \omega| + \epsilon\gamma/4.$$

Suppose $\hat{p}_t \geq \omega$. Then clearly $r_j \leq \hat{p}_t + \epsilon\gamma/4$ for $j \in J$, and thus $c(x) \leq \hat{p}_t + \epsilon\gamma/4 \leq \hat{p}_t + \gamma$ whenever $x \in C_t$. Let z be the probability that an x is chosen for which $c(x) < \hat{p}_t - \gamma$, given that x is in C_t:

$$z = \mathbf{Pr}_{x \in D}\left[c(x) < \hat{p}_t - \gamma \mid x \in C_t\right].$$

Then

$$
\begin{aligned}
p_t &= \mathbf{E}_{x \in D}\left[c(x) \mid x \in C_t\right] \\
&\leq z(\hat{p}_t - \gamma) + (1 - z)(\hat{p}_t + \epsilon\gamma/4) \\
&\leq z(p_t + \epsilon\gamma/4 - \gamma) + (1 - z)(p_t + \epsilon\gamma/2) \\
&\leq p_t + \epsilon\gamma/2 - \gamma z.
\end{aligned}
$$

This implies $z \leq \epsilon/2$, and so (3.4) holds in this case. The proof of (3.4) is symmetric when $\hat{p}_t \leq \omega$.

The algorithm of Figure 1 clearly runs in polynomial time.   ∎

It is an open question whether this class is learnable when ω is unknown.

The class of probabilistic decision lists has also been considered by Yamanishi [88]. He describes an algorithm, based on the principle of minimum description length, for learning a model of probability for p-concepts in this class; however, his algorithm is *not* computationally efficient. Also, Aiello and Mihail [1] have recently described an efficient algorithm for learning arbitrary probabilistic decision lists over the basis consisting of all literals in the special case that D is the uniform distribution.

4-3.3   Hidden-variable problems

We next consider p-concept classes motivated by *hidden-variable* problems, in which there is an underlying deterministic concept, but the settings of some of the relevant variables are invisible to the learning algorithm, resulting in apparent probabilistic behavior. A *visible monomial* p-concept is defined over $\{0,1\}^n$ by a pair (M, α), where M is a monomial over the *visible* Boolean variables $x_1, \ldots, x_n$ and $\alpha \in [0,1]$. The associated p-concept c is defined for $x \in \{0,1\}^n$ to be $c(x) = \alpha \cdot M(x)$. We conceptually regard the true deterministic concept as having the form $M \wedge I$, where I is a deterministic concept over the *hidden variables*. We interpret α as the probability that the settings of the invisible variables satisfy I. Note that we assume independence between the settings for the variables of M and those for I.

For instance, I might itself be a monomial, in which case the underlying target concept is a conjunction of literals, some which are visible and some which are hidden.

Visible monomials model well situations in which certain observable conditions are requisite to some outcome, but in which these conditions are not in themselves enough to determine the outcome with certainty. Thus, the conditions are necessary, but not sufficient, and, when the conditions are met, the final outcome may be uncertain. For instance, if you are handed a drink that is brown and fizzes and tastes sweet, then the drink might be Coke; on the other hand, it might not be Coke (it could be Pepsi). In any case, if the drink lacks any one of these qualities, then it certainly cannot be the real thing.

Theorem 3.3 *The class of visible monomial p-concepts is polynomially learnable with a model of probability.*

Proof: Let the target p-concept c be defined by the pair (M, α), and let the target distribution over $\{0,1\}^n$ be D. We describe an algorithm that, given $\epsilon, \delta > 0$, outputs with probability at least $1 - \delta$ a hypothesis h for which $\mathbf{E}_{x \in D}\left[|h(x) - c(x)|\right] \leq \epsilon$; Lemma 2.2 implies that such an algorithm can be converted into one that learns a good model of probability.

The first step of the learning algorithm is to obtain an estimate $\hat{p}$ of $p = \mathbf{Pr}_{(x,b) \in EX}\left[b = 1\right]$ that, with probability at least $1 - \delta/3$, is such that $|p - \hat{p}| \leq \epsilon/3$. If $\hat{p} \leq 2\epsilon/3$, then the algorithm outputs the hypothesis $h(x) \equiv 0$. Assuming $\hat{p}$ has the desired accuracy, we have $\mathbf{E}_{x \in D}\left[|c(x) - h(x)|\right] \leq \epsilon$ in this case as desired, since $p = \mathbf{E}_{x \in D}\left[c(x)\right] \leq \epsilon$. Otherwise, $\hat{p} > 2\epsilon/3$, and we can assume henceforth that $p \geq \epsilon/3$ (as is the case with probability at least $1 - \delta/3$).

Next our algorithm attempts to learn a good approximation of M. This is done using Valiant's algorithm [83], here denoted V, for learning monomials from positive examples only in the distribution-free deterministic model. Algorithm V, which we here use as a "black-box" subroutine, has the following properties: the algorithm takes as input positive ϵ and δ, and a source of positive examples of some monomial M, each chosen randomly according to some

fixed, arbitrary distribution D^+ on the set of all positive examples. After running for time polynomial in $1/\epsilon$, $1/\delta$ and n, V outputs a monomial $\hat{M}$ that, with high probability, has error at most ϵ for the positive examples of M, and has zero error for the negative examples. That is, with probability at least $1 - \delta$, $\hat{M}$ is such that:

$$\mathbf{Pr}_{x \in D^+}\left[\hat{M}(x) = 0\right] \leq \epsilon,$$

and also

$$M(x) = 0 \Rightarrow \hat{M}(x) = 0.$$

Our algorithm simulates V with V's parameter ϵ set to $\epsilon/4$, and δ set to $\delta/3$. We provide V with a simulated oracle EX' which supplies V with only positively labeled examples. Specifically, when V requests an example, EX' draws examples from EX until an example $(x, 1)$ is received; this instance x is then provided to V.

Note that if x is labeled positively by EX, then $c(x) > 0$ and so $M(x) = 1$. Thus, V is only supplied with positive examples. Note also that the probability of drawing a positively labeled example from EX equals p. Since $p \geq \epsilon/3$, it follows that the expected running time of EX' is at most $O(1/\epsilon)$.

The probability that EX' outputs some instance x is just

$$
\begin{aligned}
D^+(x) &= \mathbf{Pr}_{(y,b) \in EX}\left[y = x \mid b = 1\right] \\
&= \frac{\mathbf{Pr}_{(y,b) \in EX}\left[y = x \wedge b = 1\right]}{\mathbf{Pr}_{(y,b) \in EX}\left[b = 1\right]} \\
&= \frac{\alpha M(x) \cdot D(x)}{\alpha \cdot \mathbf{Pr}_{y \in D}\left[M(y) = 1\right]} \\
&= \frac{M(x)D(x)}{\mathbf{Pr}_{y \in D}\left[M(y) = 1\right]} \\
&= \mathbf{Pr}_{y \in D}\left[y = x \mid M(y) = 1\right].
\end{aligned}
$$

With probability at least $1 - \delta/3$, V outputs a hypothesis $\hat{M}$ which is such that $\hat{M}(x) = 0$ whenever $M(x) = 0$, and

$$\mathbf{Pr}_{x \in D^+}\left[\hat{M}(x) = 0\right] \leq \epsilon/4.$$

Our algorithm next obtains an estimate $\hat{\alpha}$ of $\alpha' = \mathbf{Pr}_{(x,b) \in EX}\left[b = 1 \mid \hat{M}(x) = 1\right]$ that, with probability at least $1 - \delta/3$, is such that $|\alpha' - \hat{\alpha}| \leq \epsilon/2$. Such an estimate can be derived from a polynomial-size sample since

$$\mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 1\right] \geq (1 - \epsilon/4) \cdot \mathbf{Pr}_{x \in D}\left[M(x) = 1\right] \geq (1 - \epsilon/4)p \geq (1 - \epsilon/4)(\epsilon/3).$$

The algorithm outputs the hypothesis h defined by $(\hat{M}, \hat{\alpha})$; we argue next that h is, with probability at least $1 - \delta$, within ϵ of c on average.

As noted above, $\hat{M}$ has the property that

$$\mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 0 \mid M(x) = 1\right] = \mathbf{Pr}_{x \in D^+}\left[\hat{M}(x) = 0\right] \leq \epsilon/4.$$

Also, $\hat{M}$ logically implies M. Since

$$
\begin{aligned}
\alpha &= \mathbf{Pr}_{(x,b) \in EX}\left[b = 1 \mid M(x) = 1\right] \\
&= \mathbf{Pr}_{(x,b) \in EX}\left[b = 1 \mid \hat{M}(x) = 1\right] \cdot \mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 1 \mid M(x) = 1\right],
\end{aligned}
$$

it follows that $\alpha \geq \alpha' \geq \alpha(1 - \epsilon/4) \geq \alpha - \epsilon/4$, and so $|\alpha - \hat{\alpha}| \leq 3\epsilon/4$. Thus, again making use of the fact that $\hat{M}$ has one-sided error, it can be seen that

$$
\begin{aligned}
\mathbf{E}_{x \in D}\left[|h(x) - c(x)|\right] &\leq \mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 0 \wedge M(x) = 1\right] + |\alpha - \hat{\alpha}| \cdot \mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 1\right] \\
&\leq \mathbf{Pr}_{x \in D}\left[\hat{M}(x) = 0 \mid M(x) = 1\right] + |\alpha - \hat{\alpha}| \\
&\leq \epsilon.
\end{aligned}
$$

∎

Finally, we remark that the algorithm described in this proof can be easily extended to learn any p-concept c of the form $c = \alpha c_0$ where α is an unknown constant in $[0, 1]$ and c_0 is a deterministic concept from some known concept class for which there exists an efficient algorithm that, like the algorithm V described in the proof, requires positive examples only, and outputs hypotheses with one-sided error on the positive-examples distribution only. For instance, Valiant [83] describes such an algorithm for learning k-CNF (the class of Boolean formulas consisting of a conjunction of clauses, each a disjunction of at most k literals).

4-4 Hypothesis testing and expected loss

In this section, we address the problem of *hypothesis testing* in the p-concept model. More precisely, given a labeled sample, and a hypothesis p-concept, how do we decide how good h is with respect to the sample? As will be seen, the answer to this question depends on what our goal is (a decision rule or a model of probability).

We begin with a description of the learning framework that was proposed by Haussler [36], and that extends the work of Pollard [71], Dudley [23], Vapnik [84] and others. In this framework, the learner observes pairs (x, y) drawn randomly from some product space $X \times Y_0$ according to some fixed distribution. For instance, in the p-concept model, X is the domain, and

$Y_0 = \{0, 1\}$; the target distribution on X and the target p-concept together induce a distribution on the space $X \times Y_0$.

Roughly speaking, in Haussler's model, the learner tries to find a hypothesis that accurately predicts the y-value of a random pair (x, y), given only the observed x-value. Thus, the hypothesis h should be such that $h(x)$ is "near" y for most random pairs (x, y).

It is often convenient not to restrict the range of h to the set Y_0; for instance, if $Y_0 = \{0, 1\}$, then we may want to allow h to map into $[0, 1]$. In general, then, we assume that h is a function which maps X into some set $Y \supset Y_0$.

In Haussler's model, the learner must choose a hypothesis from some given hypothesis space $\mathcal{H}$ of functions (each mapping X into Y). The goal of the learner is to find the hypothesis from $\mathcal{H}$ that minimizes the "discrepancy" on random pairs (x, y) between the observed value y, and the predicted value $h(x)$. This discrepancy between y and $h(x)$ is measured by a real-valued "loss" function. Formally, a *loss function* L is a function mapping $Y \times Y_0$ into $[0, 1]$. (The extension of such results to general bounded functions is straightforward.) Thus, the formal goal of the learner in this framework is to find a function $h \in \mathcal{H}$ that minimizes the average loss $\mathbf{E}[L(h(x), y)]$, where the expectation is over points (x, y) drawn randomly from $X \times Y_0$ according to the distribution on this product space.

Following Haussler [36], we adopt the notation $L_h(x, y) = L(h(x), y)$ for loss function L and hypothesis h. Moreover, we will write $\mathbf{E}[L_h]$ to denote the expected loss of h (with respect to L) under the unknown distribution on $X \times Y_0$. For a given sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ of m labeled examples, we will also be interested in the *empirical loss* of h:

$$\hat{\mathbf{E}}_S[L_h] = \frac{1}{m} \sum_{i=1}^{m} L_h(x_i, y_i).$$

Note that the empirical loss does not depend on the underlying distribution. Also, when the sample is clear from context, the subscript S is usually dropped.

We can cast the problems of learning decision rules and models of probability into this general framework. As mentioned above, in our setting $Y_0 = \{0, 1\}$ since an algorithm only sees $\{0, 1\}$-labels. For decision-rule learning, the algorithm outputs $\{0, 1\}$-valued hypotheses, and thus $Y = Y_0 = \{0, 1\}$ in this case. Similarly, for model-of-probability learning, we assume that hypotheses have range $[0, 1]$, and so $Y = [0, 1]$. The distribution on $X \times Y_0$ is naturally determined by the joint behavior of the target distribution D on X and the conditional probabilities $c(x)$ given by the target p-concept.

For finding the best decision rule, the *discrete* loss function is most appropriate, that is, the loss function Z given by the rule $Z(y, y') = 0$ if $y = y'$ and 1 otherwise. Then $\mathbf{E}[Z_h]$ is just the probability that h will misclassify a randomly drawn point, so minimizing $\mathbf{E}[Z_h]$ is equivalent

to minimizing the predictive error.

For finding a model of probability, the *quadratic loss function* $Q(y, y') = (y - y')^2$ has some nice properties which follow from the following theorem, and that make it the appropriate choice. Note that the empirical loss $\hat{E}[Q_h]$ is the *average squared-error* statistic that is well known to researchers in pattern recognition and statistical decision theory.

Theorem 4.1 *For any target p-concept c, target distribution D, and p-concept h,*

$$\mathbf{E}[Q_h] - \mathbf{E}[Q_c] = \mathbf{E}_{x \in D}\left[(h(x) - c(x))^2\right].$$

Proof: For fixed $x \in X$, the probability that x is labeled 1 is $c(x)$, and in this case, h has loss

$$Q_h(x, 1) = Q(h(x), 1) = (1 - h(x))^2.$$

Likewise, x is labeled 0 with probability $1 - c(x)$, and in this case, h has loss $(h(x))^2$. Thus,

$$\mathbf{E}[Q_h] = \int_X \left[c(x)(1 - h(x))^2 + (1 - c(x))h(x)^2\right] dD(x).$$

Similarly,

$$\mathbf{E}[Q_c] = \int_X \left[c(x)(1 - c(x))^2 + (1 - c(x))c(x)^2\right] dD(x).$$

Applying straightforward algebra and linearity of integrals, it follows that

$$\begin{aligned}
\mathbf{E}[Q_h] - \mathbf{E}[Q_c] &= \int_X \left[h(x) - c(x)\right]^2 dD(x) \\
&= \mathbf{E}_{x \in D}\left[(h(x) - c(x))^2\right]
\end{aligned}$$

as desired.

(All these integrals are defined, assuming as usual that c and h are measurable.)   ∎

Combined with Lemma 2.2, this theorem immediately suggests a computationally efficient method of choosing a good model of probability from a small (polynomial-size) class of candidate hypotheses. Suppose that a learning algorithm A has done some initial sampling and computation and has produced a class $\mathcal{H}$ of hypotheses, one of which is a good model of probability. Then A may simply use the empirical loss $\hat{E}[Q_h]$ on a large enough labeled sample (a second sample) as an accurate estimate of the true loss $\mathbf{E}[Q_h]$ for each $h \in \mathcal{H}$, and then output the hypothesis with the smallest empirical loss. This hypothesis h must have near minimal true loss, and so, by the preceding theorem and our assumption that $\mathcal{H}$ contains a good model of probability, h must itself be a good model of probability. The remainder of this section describes an example of an efficient learning algorithm that employs this approach.

4-4.1   Probabilistic concepts of k relevant variables

For a p-concept c on n Boolean variables, we say that variable x_i is *relevant* if $c(x) \neq c(y)$ for two vectors x and y which differ only in their ith bit. We say that c is a *p-concept of k relevant variables* if c has only k relevant variables. Such p-concepts are good models of situations in which there are a small number of variables whose settings determine the probabilistic behavior in a possibly very complicated manner, but most variables have no influence on this behavior.

Theorem 4.2 *Let $k \geq 1$ be fixed. Then the class of all p-concepts of k relevant variables is polynomially learnable with a model of probability.*

Proof: For any set $I \subset \{1, \ldots, n\}$, we say that two assignments x and y in $\{0,1\}^n$ are *equivalent with respect to I* if $x_i = y_i$ for all $i \in I$. Then this equivalence relation partitions $\{0,1\}^n$ into $2^{|I|}$ equivalence classes, called *I-blocks.*

Let c be the target p-concept, and let I_* be the set of indices of the k relevant variables of c.

Our algorithm begins by drawing a sample S_1 of size $m_1 = O((2^k/\epsilon^3) \cdot \log(2^k/\delta))$. For each of the $\binom{n}{k}$ sets I of k indices, and for each I-block B, our algorithm obtains from S_1 an estimate $\hat{p}_B$ of $p_B = \mathbf{Pr}_{(x,b) \in EX}[b = 1 \mid x \in B]$. A hypothesis h_I is then defined by the rule $h_I(x) = \hat{p}_B$ for $x \in B$.

By our choice of m_1, it follows from Chernoff bounds (Lemma 2-3.6) that, with probability at least $1 - \delta/2$, a sample S_1 is chosen for which $|\hat{p}_B - p_B| \leq \epsilon/4$ for every I_*-block B which satisfies $\mathbf{Pr}_{x \in D}[x \in B] > \epsilon/2^{k+2}$. This implies that, with high probability,

$$\mathbf{E}_{x \in D}[|h_{I_*}(x) - c(x)|] = \sum_B \mathbf{Pr}_{x \in D}[x \in B] \cdot |\hat{p}_B - c(x)| \leq \epsilon/2$$

where the sum is taken over all I_*-blocks B. This bound follows from the fact that $c(x) = p_B$ for $x \in B$, and by breaking the sum into two parts according to whether $\mathbf{Pr}_{x \in D}[x \in B]$ exceeds or does not exceed $\epsilon/2^{k+2}$.

Next, our algorithm tests each hypothesis h_I; that is, an estimate $\hat{\mathbf{E}}[Q_{h_I}]$ is found from a sufficiently large sample S_2 that, with high probability, is within $\epsilon/4$ of $\mathbf{E}[Q_{h_I}]$. Specifically, this will be the case with probability at least $1 - \delta/2$ for all hypotheses h_I if we choose a sample S_2 of size $O((1/\epsilon^2) \cdot \log(n^k/\delta))$. The algorithm outputs the hypothesis $h = h_I$ with the minimum empirical loss. Then, applying Theorem 4.1, we have:

$$\begin{aligned}
\mathbf{E}_{x \in D}[(h(x) - c(x))^2] &= \mathbf{E}[Q_h] - \mathbf{E}[Q_c] \\
&\leq \hat{\mathbf{E}}[Q_h] - \mathbf{E}[Q_c] + \epsilon/4 \\
&\leq \hat{\mathbf{E}}[Q_{h_{I_*}}] - \mathbf{E}[Q_c] + \epsilon/4
\end{aligned}$$

$$\leq \quad E[Q_{h_{l_*}}] - E[Q_c] + \epsilon/2$$

$$= \quad E_{x \in D}\left[(h_{l_*}(x) - c(x))^2\right] + \epsilon/2 \leq \epsilon.$$

Applying Lemma 2.2, it follows that this efficient algorithm can be used to learn a good model of probability. ∎

4-5 Uniform convergence methods

When is minimization of the empirical loss over a hypothesis class $\mathcal{H}$ sufficient to insure good learning of a decision rule or a model of probability? Note that even with computational issues set aside, the hypothesis-testing methods of the preceding section fall apart in the case of an infinite class $\mathcal{H}$: directly estimating the empirical loss of each $h \in \mathcal{H}$ separately would take an infinite number of examples and an infinite amount of time. What is required is a characterization of the number of examples required for uniform convergence of empirical losses to expected losses analogous to that provided by the VC-dimension in the case of deterministic concepts. This is particularly pressing in our model of p-concepts, where even when the domain is finite (e.g. $\{0,1\}^n$), the target p-concept class is usually infinite due to the different values allowed for the probabilities. We now turn to a discussion of such uniform convergence techniques applicable to p-concept classes.

Haussler [36], Pollard [71] and others have described the *combinatorial dimension* of a class of real-valued functions $\mathcal{F}$, and have shown that the combinatorial dimension is a powerful tool for obtaining uniform convergence results. Before defining the combinatorial dimension, we state its most important property for us, namely, that it allows us to upper bound the size of a sample sufficient to guarantee uniform convergence of empirical estimates for the entire class of functions. The following theorem is adapted directly from Haussler's Corollary 2 [35].

For a hypothesis space $\mathcal{H}$ and loss function L, we define $L_{\mathcal{H}} = \{L_h : h \in \mathcal{H}\}$.

Theorem 5.1 *Let $\mathcal{H}$ be a hypothesis space of functions mapping X into Y which satisfies certain "permissibility" assumptions (see Haussler's paper). Let D be a probability distribution on $X \times Y_0$, let $L : Y \times Y_0 \to [0,1]$ be a loss function, let $d < \infty$ be the combinatorial dimension of $L_{\mathcal{H}}$, and let S be a sample of m points from $X \times Y_0$ chosen randomly according to D. Assume*

$$m \geq m(d, \epsilon, \delta) = \frac{72}{\epsilon^2}\left(2d\ln\left(\frac{24e}{\epsilon}\right) + \ln\left(\frac{8}{\delta}\right)\right).$$

Then

$$\Pr[\exists h \in \mathcal{H} : \left|\hat{E}[L_h] - E[L_h]\right| > \epsilon] \leq \delta,$$

where the probability is taken over the random generation of S according to D.

Theorem 5.1 suggests the following canonical algorithm for finding a hypothesis from $\mathcal{H}$ with near minimum loss, when the combinatorial dimension d is finite: take a sample S of at least $m(d, \epsilon/2, \delta)$ labeled examples from the oracle EX, and output any $h \in \mathcal{H}$ that minimizes the empirical loss $\hat{\mathbf{E}}[L_h]$ with respect to S. Then the theorem guarantees that the output hypothesis has true loss within ϵ of the best possible with probability at least $1 - \delta$. This of course ignores the computational problem of actually finding such a hypothesis.

We can apply Theorem 5.1 to our learning problems by determining what the combinatorial dimension is for each of the loss functions Z and Q. For the loss function Z, Haussler points out that the combinatorial dimension is just the VC-dimension of the hypothesis class. That is, if $\mathcal{H}$ is a hypothesis space of functions with range $\{0, 1\}$, then the combinatorial dimension of the set of functions $Z_{\mathcal{H}}$ is just the VC-dimension of $\mathcal{H}$. Thus, the number of examples needed for decision-rule learning is bounded by the VC-dimension of the space of hypotheses used by the learning algorithm. (That the VC-dimension can be used in this manner was also observed by Blumer et al. [14].)

For example, consider the problem of learning a decision rule for an increasing function over $\mathbb{R}$. Note that the best decision rule for such a p-concept is always of the form $h_a(x) = 1$ for $x > a$, and 0 otherwise, for some a. Thus, a natural and efficient decision-rule learning algorithm for this problem is the following: draw a "large" sample from EX. Then, for each x_i in the sample, determine the empirical predictive error of hypothesis h_{x_i}, that is, the fraction of points in the sample whose labels disagree with h_{x_i}. Finally, output the h_{x_i} with the minimum empirical predictive error. Since the VC-dimension of this class of decision rules is 1, it follows from Theorem 5.1 that a polynomial-size sample suffices to insure the correctness of this algorithm.

For the problem of learning a model of probability, we introduce the *quadratic loss dimension*. Let $\mathcal{H}$ be a class of p-concepts over domain X. Let $T = \{(x_1, r_1), \ldots, (x_d, r_d)\}$ be a set of d pairs, where each $x_i \in X$ and each $r_i \in [0, 1]$. We say that $\mathcal{H}$ *shatters* T if for every string $v \in \{0, 1\}^d$ there is a p-concept $h \in \mathcal{H}$ such that for $1 \leq i \leq d$, if $v_i = 0$ then $h(x_i) \leq r_i$ and if $v_i = 1$ then $h(x_i) > r_i$. Thus on the points $x_1, \ldots, x_d$ the class $\mathcal{H}$ exhibits all 2^d possible "above-below" behaviors with respect to the r_i. A geometric interpretation of this definition is to regard $(r_1, \ldots, r_d)$ as the origin of a coordinate system in d-dimensional Euclidean space; then $\mathcal{H}$ shatters T if the set $\{(h(x_1), \ldots, h(x_d)) : h \in \mathcal{H}\}$ intersects all 2^d orthants of the coordinate system. For this reason we will sometimes refer to $(r_1, \ldots, r_d)$ as the *origin of shattering*. The *quadratic loss dimension* of a p-concept class $\mathcal{H}$, denoted $QD(\mathcal{H})$, is defined as the largest value of d such that there is some set T of d pairs that is shattered by $\mathcal{H}$; if no such d exists, then $QD(\mathcal{H})$ is infinite.

The quantity $QD(\mathcal{H})$ is in fact just the combinatorial dimension of the class $\mathcal{H}$; but since

we evaluate hypotheses via the quadratic loss, we are more concerned with the combinatorial dimension of the associated class $Q_{\mathcal{H}}$ of loss functions. The following theorem states that the combinatorial dimension of $Q_{\mathcal{H}}$ is also equal to $QD(\mathcal{H})$. Despite these equivalences, we choose to use the notation $QD(\mathcal{H})$ to emphasize that in more general settings, the combinatorial dimension of the class of loss functions may be quite different than that of the underlying hypothesis class $\mathcal{H}$.

Theorem 5.2 *For any p-concept class $\mathcal{H}$, the combinatorial dimension of $Q_{\mathcal{H}}$ is equal to the quadratic loss dimension of $\mathcal{H}$.*

Proof: Let $\{(x_i, r_i)\}_{i=1}^d$ shatter $\mathcal{H}$. For all $v \in \{0,1\}^d$, there exists $h \in \mathcal{H}$ such that

$$sign(r_i - h(x_i)) = v_i,$$

where $sign(y) = 1$ if $y \geq 0$ and $sign(y) = 0$ if $y < 0$. Since all quantities are non-negative, $h(x_i) \leq r_i$ if and only if $Q_h(x_i, 0) = (h(x_i))^2 \leq r_i^2$. Thus,

$$v_i = sign(r_i^2 - Q_h(x_i, 0)),$$

and so $\{((x_i, 0), r_i)\}_{i=1}^d$ shatters $Q_{\mathcal{H}}$. Thus, the combinatorial dimension of $Q_{\mathcal{H}}$ is at least $QD(\mathcal{H})$.

Conversely, let $\{((x_i, b_i), r_i)\}_{i=1}^d$ shatter $Q_{\mathcal{H}}$. Since d is finite, we can assume without loss of generality that the r_i's are chosen so that strict inequality holds in the definition of combinatorial dimension, i.e., for all $v \in \{0,1\}^d$ there exists $h \in \mathcal{H}$ such that $Q_h(x_i, b_i) < r_i$ if $v_i = 1$ and $Q_h(x_i, b_i) > r_i$ if $v_i = 0$. Then

$$sign(r_i - Q_h(x_i, b_i)) = sign(r_i - (h(x_i) - b_i)^2) = sign(\sqrt{r_i} - |h(x_i) - b_i|)$$

which equals $sign(\sqrt{r_i} - h(x_i))$ if $b_i = 0$, and equals $sign(h(x_i) - (1 - \sqrt{r_i})) = 1 - sign((1 - \sqrt{r_i}) - h(x_i))$ if $b_i = 1$. It follows that $\{(x_i, |b_i - \sqrt{r_i}|)\}_{i=1}^d$ shatters $\mathcal{H}$, and thus the combinatorial dimension of $Q_{\mathcal{H}}$ is at most $QD(\mathcal{H})$. ∎

Note that the second part of the proof of this theorem relies critically on the fact that, in the p-concept model, instances are only $\{0, 1\}$-labeled.

4-5.1   Linear function spaces

Armed with the definition of quadratic loss dimension and the sample size upper bounds provided by Theorem 5.1, we can now seek efficient algorithms that work by directly minimizing the quadratic loss over an infinite class of functions. This is the approach taken in our next

theorem. For any domain X, let $f_i : X \to \mathbb{R}, 1 \le i \le d$ be any d functions, and let $C(f_1, \ldots, f_d)$ denote the class of all p-concepts of the form $c(x) = \sum_{i=1}^{d} a_i f_i(x)$ for $a_i \in \mathbb{R}$, where we assume that the f_i and a_i are such that $c(x) \in [0,1]$ for all $x \in X$. We describe below an algorithm that learns a model of probability for p-concepts in the class $C(f_1, \ldots, f_d)$. The running time of this algorithm is polynomial in the usual parameters, d, and the time needed to evaluate the functions f_i.

This result can be applied to prove the polynomial learnability of several natural p-concept classes. For instance, consider the generalization of deterministic disjunctions in which the target p-concept has the form $c(x) = (x_{i_1} + \cdots + x_{i_t})/t$, where the x_{i_j} are Boolean variables chosen from $x_1, \ldots, x_n$, and $+$ denotes ordinary addition. Thus, such a p-concept is "more positive" on vectors $x \in \{0,1\}^n$ that have many of the relevant variables set to 1. Such a p-concept class is clearly of the form required by Theorem 5.3, and so is polynomially learnable with a model of probability.

As a more subtle application, consider a class of p-concepts over $\{0,1\}^n$ that are partially specified by a canonical positive example $z \in \{0,1\}^n$. We wish to model a setting in which z is the prototypical positive instance, and those examples "most like" z are more likely to be labeled positively. Thus, the target p-concept might have the form $c(x) = a - b \cdot d(x, z)$ where $d(x, z)$ denotes the Hamming distance and a and b are positive real-valued coefficients such that c is maximized at z and is always in the range $[0, 1]$. Here the p-concept class C is obtained by ranging over the choices of the prototype z and the coefficients a and b, and the "decay function," which specifies the rate at which vectors further away from the prototype fail to exemplify the concept, is linear. It is not difficult to show that each function in C can in fact be written as a weighted linear sum of the variables $x_1, \ldots, x_n$, so C is polynomially learnable with a model of probability.

Finally, we remark that Theorem 5.3 can be applied to learn so-called "t-transform functions" considered by Mansour [60].

Theorem 5.3 *For any set of d known computable functions $f_i : X \to \mathbb{R}, 1 \le i \le d$, the class $C(f_1, \ldots, f_d)$ is learnable with a model of probability. Specifically, there exists a learning algorithm for this class whose running time is polynomial in $1/\epsilon$, $1/\delta$, $1/\gamma$, d and the maximum time needed to evaluate any of the functions f_i.*

Proof: Given $\epsilon, \delta > 0$, our algorithm draws a sample of size $m = \lceil m(d, \epsilon/2, \delta) \rceil$ as given by Theorem 5.1, and attempts to find the choice of coefficients $a_1, \ldots, a_d$ that minimizes the quadratic loss over the sample. This can be done using a standard least-squares approximation. For instance, this can be done directly by differentiating with respect to each unknown coefficient a_i the expression $\sum_{j=1}^{m} \left[\left(\sum_{i=1}^{d} a_i f_i(x_j) \right) - b_j \right]^2$ (where $\{(x_j, b_j)\}_{j=1}^{m}$ is the labeled sample), and

setting the resulting partial derivative to 0. This yields a system of d linear equations in the d variables a_i that is of a special form and that can be solved using standard techniques. Cormen, Leiserson and Rivest [19, Chapter 31] describe in detail how this can be done efficiently; see also Duda and Hart [22].

Let $\hat{a}_1, \ldots, \hat{a}_d$ be the resulting solution, and let $h_0 = \sum_{i=1}^d \hat{a}_i f_i$. Note that h_0 may not be bounded between 0 and 1, and so may not be in $C = C(f_1, \ldots, f_d)$. We show below how to handle this difficulty.

For any real-valued function f, let $clamp(f)$ denote the function obtained by "clamping" f between 0 and 1; that is, $clamp(f) = g \circ f$ where $g : \mathbb{R} \to \mathbb{R}$, and $g(x)$ is defined to be 0 if $x \leq 0$, x if $0 \leq x \leq 1$, and 1 if $x \geq 1$. Let $\mathcal{H} = \left\{ clamp\left(\sum_{i=1}^d a_i f_i\right) : a_i \in \mathbb{R} \right\}$. Our algorithm outputs the hypothesis $h = clamp(h_0)$. Clearly h is in $\mathcal{H}$, as is the target c.

Dudley [23] shows that a d-dimensional linear function space has combinatorial dimension d. (This is reproved by Haussler [35], Theorem 4.) Combined with Haussler's Theorem 5 (which concerns the combinatorial dimension of families of functions constructed in the same way as $\mathcal{H}$), this immediately implies $QD(\mathcal{H}) \leq d$. Thus, by Theorem 5.1 and our choice of m, with probability at least $1 - \delta$, $\left| E[Q_{h'}] - \hat{E}[Q_{h'}] \right| \leq \epsilon/2$ for every $h' \in \mathcal{H}$. Also, note that $\hat{E}[Q_h] \leq \hat{E}[Q_{h_0}]$ since all instances in the sample are $\{0, 1\}$-labeled, so clamping the hypothesis only improves its performance. Thus, with probability at least $1 - \delta$, we have:

$$
\begin{aligned}
E_{x \in D}\left[(h(x) - c(x))^2\right] &= E[Q_h] - E[Q_c] \\
&\leq \hat{E}[Q_h] - E[Q_c] + \epsilon/2 \\
&\leq \hat{E}[Q_{h_0}] - E[Q_c] + \epsilon/2 \\
&\leq \hat{E}[Q_c] - E[Q_c] + \epsilon/2 \leq \epsilon.
\end{aligned}
$$

As usual, Lemma 2.2 can be applied to convert this algorithm into one that learns a good model of probability for this class.   ∎

4-6   A lower bound on sample size

Theorem 5.1 provides a kind of general upper bound on the sample size required for learning a model of probability. We turn now to the problem of lower bounds on sample complexity in this framework. For this, we need to introduce a refined notion of shattering.

Let $\mathcal{H}$ be a class of p-concepts over domain X. Let $T = \{(x_1, r_1), \ldots, (x_d, r_d)\}$ be a set of d pairs, where each $x_i \in X$ and each $r_i \in [0, 1]$. For $w > 0$, we say that $\mathcal{H}$ w-shatters T if for every string $v \in \{0, 1\}^d$ there is a p concept $h \in \mathcal{H}$ such that for $1 \leq i \leq d$, if $v_i = 0$ then $h(x_i) < r_i - w$ and if $v_i = 1$ then $h(x_i) > r_i + w$. Thus, in addition to T being shattered by

$\mathcal{H}$ we require that there be a *separation* of width w between r_i and each $h(x_i)$; we call w the *width of shattering*. Note that if $\mathcal{H}$ has quadratic dimension at least d then there always exists some $w > 0$ such that some set of d pairs over $X \times [0,1]$ is w-shattered.

Based on this stronger notion of shattering, we can now prove the following lower bound on sample complexity in our model. This lower bound, combined with Theorems 5.1 and 5.2, shows that when the quadratic loss dimension is finite it characterizes the sample size required for learning with a model of probability (that is, the bound obtained by applying Theorem 5.1 is tight within a polynomial factor of $1/\epsilon$ and $1/\delta$). This lower bound may also be of theoretical interest, since in Haussler's general learning framework [36] the combinatorial dimension is only an approximate upper bound on the so-called *covering number*, which is directly used to obtain sample-size bounds.

Theorem 6.1 *Let C be a p-concept class that w-shatters a set of cardinality d. Then for $\gamma \le w$ and $\epsilon + \delta \le 1/8$, any algorithm for learning C with a model of probability requires at least $\lfloor d(\lg e)/8 \rfloor = \Omega(d)$ examples.*

Proof: Our proof is based on the analogous lower bound proof given by Blumer et al. [14] for learning deterministic concepts. However, the analysis is more involved in the probabilistic case.

Let $T = \{(x_1, r_1), \ldots, (x_d, r_d)\}$ be w-shattered by the p-concept class C. Let $C_0 \subseteq C$ be any fixed subclass of C such that C_0 w-shatters T and $|C_0| = 2^d$. Let A be a learning algorithm for C taking m examples for the given choices of ϵ, δ and γ, and let h_S denote the hypothesis output by A on input a labeled sample S of size m. (We assume for simplicity that A is deterministic — the proof is easily modified to handle randomized algorithms.)

Let the target distribution D be uniform over the points $x_1, \ldots, x_d$. We define a weak error measure $e(c, S)$ for target $c \in C_0$ and input sample S as follows: the error $e_i(c, S)$ at x_i is defined to be 0 if $c(x_i)$ and $h_S(x_i)$ are either both less than r_i, or both greater than r_i; otherwise, $e_i(c, S) = 1$. Then e is just the average of the e_i's:

$$e(c, S) = \frac{1}{d} \sum_{i=1}^{d} e_i(c, S).$$

Note that if $e(c, S) > \epsilon$, then h_S cannot be an (ϵ, w)-good model of probability for c since if $c(x_i)$ and $h_S(x_i)$ are not "on the same side" of r_i, then they differ by more than w.

This definition allows us to examine the expectation

$$\mathbf{E}_S[e(c, S)] = \sum_S \mathbf{Pr}[S|c] \cdot e(c, S)$$

which is taken over S drawn randomly according to D and labeled randomly according to c, and $\mathbf{Pr}[S|c]$ is the conditional probability that S is generated by D and c.

We will also be interested in the expectation of $e(c, S)$ when both $c \in C_0$ is generated uniformly at random and S is generated according to the randomly chosen c and the target distribution D:

$$\mathbf{E}_{c,S}[e(c, S)] = \frac{1}{2^d} \sum_S \sum_{c \in C_0} \mathbf{Pr}[S|c] \cdot e(c, S).$$

We wish to lower bound $e(c, S)$ for most of the p-concepts in C_0. For any sample S, let $C_S = \{c \in C_0 : e(c, S) < 1/4\}$. Then for $c \in C_0 - C_S$, $e(c, S) \geq 1/4$. and so we obtain the lower bound

$$\mathbf{E}_{c,S}[e(c, S)] \geq \frac{1}{2^{d+2}} \sum_S \sum_{c \in C_0 - C_S} \mathbf{Pr}[S|c] = \frac{1}{2^{d+2}} \left(\sum_S \sum_{c \in C_0} \mathbf{Pr}[S|c] - \sum_S \sum_{c \in C_S} \mathbf{Pr}[S|c] \right).$$

Now

$$\sum_S \sum_{c \in C_0} \mathbf{Pr}[S|c] = \sum_{c \in C_0} \sum_S \mathbf{Pr}[S|c] = |C_0| = 2^d$$

since for any c, $\sum_S \mathbf{Pr}[S|c] = 1$. For the second term of the expectation, we will upper bound the total number of possible samples S, the cardinality $|C_S|$, and the value of $\mathbf{Pr}[S|c]$. First, the number of possible samples S is at most $(2d)^m$, since each of the d points may appear with either label and the number of examples in S is m. To bound $|C_S|$, consider drawing a p-concept c uniformly at random from the class C_0. By choice of C_0, the probability that $e(c, S) < 1/4$ is bounded by the probability of fewer than $d/4$ heads occurring in d flips of a fair coin. Thus. applying Chernoff bounds (Lemma 2-3.6), we conclude

$$|C_S| \leq |C_0| \cdot e^{-d/8} = 2^{(1-a_0)d}$$

where $a_0 = (\lg e)/8$. Finally, $\mathbf{Pr}[S|c] \leq 1/d^m$ since if we ignore the labels on the points in S, the probability of any particular set of m points being generated by the target distribution D is at most $1/d^m$.

Piecing together these bounds. we may now write

$$\mathbf{E}_{c,S}[e(c, S)] \geq \frac{1}{2^{d+2}} \left(2^d - (2d)^m \cdot 2^{(1-a_0)d} \cdot d^{-m} \right) = \frac{1}{4} \left(1 - 2^{m-a_0 d} \right).$$

Thus, if $m \leq a_0 d - 1$ then $\mathbf{E}_{c,S}[e(c, S)] \geq 1/8$. From this it follows that for some fixed $c_0 \in C_0$, $\mathbf{E}_S[e(c_0, S)] \geq 1/8$ where the expectation is taken over S drawn according to D and labeled according to c_0. By assumption A learns with a model of probability. Thus, with probability at least $1 - \delta$, a sample S is chosen such that h_S is an (ϵ, w)-good model of probability. As noted above, in such a case, $e(c, S) \leq \epsilon$. Thus, $\mathbf{E}_S[e(c_0, S)] \leq (1 - \delta)\epsilon + \delta < \epsilon + \delta$. Therefore,

if $\epsilon + \delta \le 1/8$ then m is at least $\lfloor a_0 d \rfloor$, proving the theorem.   ∎

Any theorem giving a sample-size lower bound must incorporate the width of shattering; for instance, ours holds only for $\gamma \le w$. To see that this is necessary, note that the p-concept class of all functions mapping X into $\{1/2 - w, 1/2 + w\}$ shatters all of X, but for $\gamma \ge w$ this class can be learned with *no* examples with the hypothesis $h(x) \equiv 1/2$. A more natural example is provided by the non-decreasing functions of Section 4-3.1. Here the quadratic loss dimension is infinite, but we have an efficient learning algorithm. An interesting open problem is to give improved general upper bounds on sample size that incorporate the width of shattering.

4-7   Occam's Razor for general loss functions

In this section, we present a generalized form of Occam's Razor [13] applicable to the minimization of bounded loss functions, and in particular to learning p-concepts with a model of probability or a decision rule. Here we have several motivations: first, it is of philosophical interest to investigate the most general conditions under which learning is equivalent to some form of data compression; second, as in the Valiant model, we hope that Occam's Razor will help isolate and simplify the probabilistic analysis of learning algorithms; third, Occam's Razor may be easier to apply than uniform-convergence methods in the case that the combinatorial dimension is unknown or difficult to compute; and fourth, Occam's Razor may give better sample-size bounds than direct analyses.

An *Occam algorithm* for hypothesis class $\mathcal{H}$ over parametrized domain X, with respect to a loss function $L : Y \times Y_0 \to [0, 1]$ is a polynomial-time algorithm A that takes as input a labeled sample $S \in (X_n \times Y_0)^m$, and outputs a hypothesis h with the properties that:

1. $\hat{\mathbf{E}}[L_h] - \inf_{h' \in \mathcal{H}} \hat{\mathbf{E}}[L_{h'}] \le \tau = \tau(n, m) = n^a m^{-\alpha}$ for some constants $a \ge 0$, and $\alpha > 0$; and

2. h can be represented by a string over the finite alphabet $\{0, 1\}$ of encoded length $\ell = \ell(n, m) = n^b m^\beta$ for some constants $b \ge 0$ and $\beta < 1$.

Thus, as in the non-probabilistic setting, we require an Occam algorithm to perform some kind of data compression, i.e., to output a hypothesis significantly smaller than the given sample. Furthermore, the output hypothesis must come close to minimizing the empirical loss on the sample over the entire hypothesis space $\mathcal{H}$.

Theorem 7.1 *Let A be an Occam algorithm as described above. Let S be a labeled sample of size m generated according to some target p-concept c. Let h be the result of running A on S.*

Assume m is so large that $\tau \le \epsilon/4$ and $2(2^\ell + 1)e^{-\epsilon^2 m/8} \le \delta$. Then

$$\Pr[\mathrm{E}[L_h] - \inf_{h' \in \mathcal{H}} \mathrm{E}[L_{h'}] > \epsilon] \le \delta.$$

In particular, this will be the case if

$$m \ge \max\left\{ \left(\frac{4n^a}{\epsilon}\right)^{1/\alpha}, \left(\frac{16(\ln 2)n^b}{\epsilon^2}\right)^{1/(1-\beta)}, \frac{16\ln(4/\delta)}{\epsilon^2} \right\}.$$

Proof: The proof is analogous to that of Blumer et al. [13].

Let $\mathcal{H}_A$ be the set of (at most) 2^ℓ hypotheses which might potentially be output by A. Let $h_* \in \mathcal{H}$ be such that $\mathrm{E}[L_{h_*}] \le \inf_{h' \in \mathcal{H}} \mathrm{E}[L_{h'}] + \epsilon/4$. Then, by Chernoff bounds (Lemma 2-3.6), the probability that either $\mathrm{E}[L_{h'}] > \hat{\mathrm{E}}[L_{h'}] + \epsilon/4$ for any $h' \in \mathcal{H}_A$, or that $\hat{\mathrm{E}}[L_{h_*}] > \mathrm{E}[L_{h_*}] + \epsilon/4$ is at most $2(2^\ell + 1)e^{-\epsilon^2 m/8} \le \delta$. So, with probability at least $1 - \delta$,

$$
\begin{aligned}
\mathrm{E}[L_h] &\le \hat{\mathrm{E}}[L_h] + \epsilon/4 \\
&\le \inf_{h' \in \mathcal{H}} \hat{\mathrm{E}}[L_{h'}] + \epsilon/2 \\
&\le \hat{\mathrm{E}}[L_{h_*}] + \epsilon/2 \\
&\le \mathrm{E}[L_{h_*}] + 3\epsilon/4 \\
&\le \inf_{h' \in \mathcal{H}} \mathrm{E}[L_{h'}] + \epsilon.
\end{aligned}
$$

We show next that the stated bound on m is sufficient. Clearly, from the first bound on m, $\tau \le \epsilon/4$. Further, from the second bound, we have that $\ell = n^b m^\beta \le (\lg e)\epsilon^2 m/16$. Thus, $2(2^\ell + 1)e^{-\epsilon^2 m/8} \le 4 \cdot 2^\ell e^{-\epsilon^2 m/8} \le 4 \cdot e^{-\epsilon^2 m/16} \le \delta$ by the last bound on m. ∎

As an example, Theorem 7.1 can be applied to the problem of learning p-concepts with k relevant variables. Essentially, the algorithm given in Theorem 4.2 can be modified so that a single initial sample of size m can be used for all of the estimates made by that algorithm. Note that a hypothesis output by this algorithm can be represented by the names of k of the variables, plus the probabilities for the 2^k equivalence classes. Each name requires $\lg n$ bits, and moreover, each probability is a rational number (being an empirical probability estimate) that requires only $O(\log m)$ bits; thus, the hypothesis has size $O(k \log n + 2^k \log m)$. Finally, it can be shown that the hypothesis has the minimum empirical loss over the *entire* class of p-concepts with k relevant variables. Thus, Theorem 7.1 can be used to easily determine an appropriate sample size for this algorithm.

Note that Theorem 7.1 is only applicable to algorithms which output hypotheses over a finite alphabet. However, the theorem can be extended to apply to other algorithms in a manner similar to the approach taken by Littlestone and Warmuth [59] in the Valiant model. (See

also Section 2-6.2.) The basic idea is to allow the learning algorithm to output hypotheses that can be represented over the alphabet $S \cup \{0, 1\}$, where S is the given sample. That is, the representation of the hypothesis may include individual examples from the sample itself. For example, the hypothesis output by the algorithm for learning increasing functions with a decision rule (Section 4-5) can be represented by a single example from the sample, despite the fact that this hypothesis would require an infinite number of bits to represent over a fixed finite alphabet. Thus, this alternate form of Occam's Razor can be used to provide a good sample-size bound. Similarly, the algorithm given in Theorem 3.1 (slightly modified) for learning increasing functions with a model of probability can be cast in this light as an Occam algorithm.

4-8   Conclusions and open problems

In this chapter, we have explored an extension of Valiant's model that incorporates the uncertainty inherent to many real-world learning problems. We have focused primarily on techniques for the design of efficient algorithms in this model.

Naturally, we would like to find efficient algorithms for much broader classes of p-concepts than the simple classes considered here. For example, can the algorithm of Section 4-3.2 be extended to learn arbitrary (not necessarily ω-converging) probabilistic decision lists? As is often the case in the deterministic Valiant model, sample size is not the problem: from Theorem 7.1, one can fairly easily derive a polynomial sample-size bound for learning this class using a computationally inefficient Occam algorithm that, given a sample, finds the decision list with the minimum quadratic loss by trying all permutations of the list order. The problem here is computational: how can we learn this class efficiently? The development of further techniques for learning p-concepts is a vitally important direction for further research.

Although the p-concept model captures realistic aspects of many learning problems, it might still be criticized for its assumption that the target p-concept belongs to an a priori known class of p-concepts. More realistic is a so-called *agnostic* learning model in which the target p-concept is *any* function from X into $[0, 1]$, and the learner's goal is to find the best hypothesis from some fixed space of hypotheses. This is actually the framework assumed by Haussler [36] in deriving his sample-size bounds. A few of the algorithms described in this chapter are effective agnostic learners, such as the algorithm of Theorem 4.2 for p-concepts with k relevant variables. An important open problem is the extension of other algorithms to agnostic learning. For instance, do there exist efficient agnostic algorithms for probabilistic decision lists with ω-converging probabilities (Section 4-3.2), or for linear function spaces (Section 4-5.1)?

It is also important to continue to develop a theoretical foundation for p-concept learning. For instance, are there other loss functions that might be appropriate, such as the log loss function? (See Haussler [37] in this regard.) Also, can the lower bound proof of Theorem 6.1

be significantly improved?

Finally, consistent with our quest for efficient algorithms is the need to be able to recognize that a learning problem is computationally intractable. Various techniques in this regard have been developed in the Valiant model, such as those of Pitt and Valiant [68], and Kearns and Valiant [52, 49]. Can such techniques be extended to the p-concept model? Both of these results seem to depend crucially on the deterministic nature of the Valiant model. What then would a negative, computational result look like in the p-concept model?

Inference of Finite Automata Using Homing Sequences

5-1 Introduction

Imagine a simple, autonomous robot placed in an unfamiliar environment. Typically, such a robot would be equipped with some sensors (a camera, sonar, a microphone, etc.) that provide the robot limited information about the state of its environment. Being autonomous, the robot would also have some simple actions that it has the option of executing (step ahead, turn left, lift arm, etc.).

For instance, the robot might be in the simple toy environment of Figure 1. In this environment, the robot can sense its local environment (whether the "room" it occupies is shaded or not), and can traverse one of the out-going edges by executing action "x" or action "y."

A priori, the robot may not be aware of the "meaning" of its actions, nor of the sense data it is receiving. It may also have little or no knowledge beforehand about the "structure" of its environment.

This problem motivates the research presented in this chapter: how can the robot infer on its own from experience a good model of its world? Specifically, such a model should explain and predict how the robot's actions affect the sense data received.

Certainly, once such a model has been inferred, the robot can function more effectively in the learned environment. However, programming the robot with a complete model of a fairly complex environment would be prohibitively difficult; what's more, even if feasible, a robot with a pre-programmed world model is entirely lacking in flexibility and would likely have a hard time coping in environments other than the one for which it was programmed. Thus, the development of effective learning methods would both simplify the job of the programmer, and
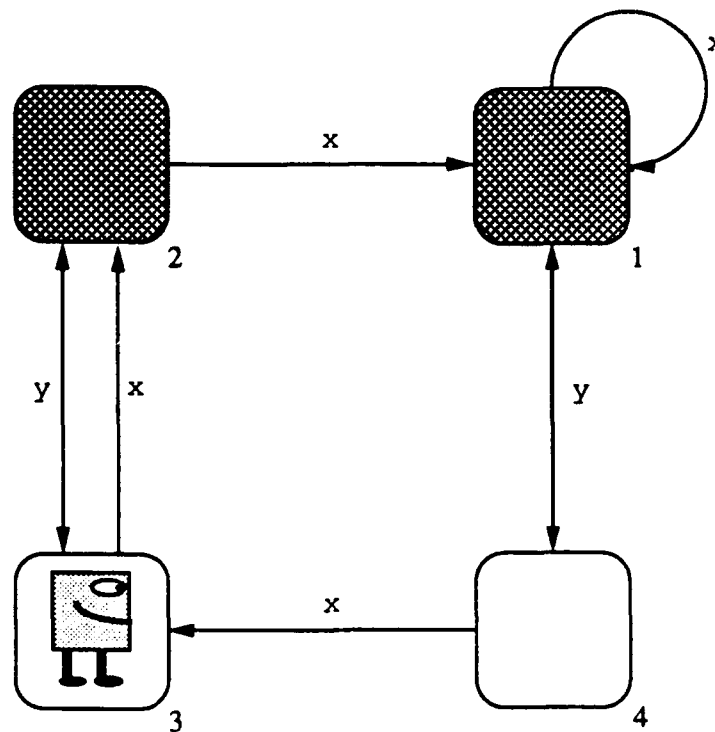
Figure 1: An example robot environment.

make for a more versatile robot.

This problem of learning about a new environment from experience has been addressed by a number of researchers using a variety of approaches: Drescher [21] explores learning in quite rich environments using an approach based on Piaget's theories of early childhood development. Wilson [87] studies so-called genetic algorithms for learning by "animats" in unfamiliar environments. Kuipers and Byun [56] advocate a "qualitative" approach to the related problem of learning a map of a mobile robot's environment. The map-learning problem is also studied by Mataric [61].

In this chapter, we take an initial step toward a general, algorithmic solution to the robot's learning problem. Specifically, we give a thorough treatment to the problem of inferring the structure of an environment that is known a priori to be *deterministic* and *finite state*. Such an environment can be naturally modeled as a deterministic finite-state automaton: the robot's actions then are the inputs to the automaton, and the automaton's output is just the sense data the robot receives from the environment. Our goal then is to infer the unknown automaton by observing its input-output behavior.

This problem has been well studied by the theoretical community, and it continues to

generate new interest. (See Pitt's paper [67] for an excellent survey.) Virtually all previous research, however, has assumed that the learner has a means of "resetting" the automaton to some start state. Such an assumption is quite unnatural, given our motivation; as in real life, we expect the robot to learn about its environment in one continuous experiment. The main result of this chapter is the first set of provably effective algorithms for inferring finite-state automata in the absence of a reset.

Here is a brief history of some of the previous, theoretical work on inference of automata. The most important lesson of this research has been that a combination of active experimentation and passive observation is both necessary and sufficient to learn an unknown automaton.

Angluin [3] and Gold [29] show that it is NP-complete to find the smallest automaton consistent with a given sample of input-output pairs. Pitt and Warmuth [69] show that merely finding an approximate solution is intractable (assuming $P \neq NP$). In the Valiant model, Kearns and Valiant [52] consider the problem of predicting the output of the automaton on a randomly chosen input, based on a random sample of the machine's behavior. Extending the work of Pitt and Warmuth [70], they show that this problem is intractable, assuming the security of various cryptographic schemes. Thus, learning by passively observing the behavior of the unknown machine is apparently infeasible.

What about learning by actively *experimenting* with it? Angluin [5] shows that this problem is also hard. She describes a family of automata which cannot be identified in less than exponential time when the learner can only observe the behavior of the machine on inputs of the learner's own choosing. The difficulty here is in accessing certain hard-to-reach states.

In spite of these negative results, Angluin [6], elaborating on Gold's results [28], shows that a *combination* of active and passive learning is feasible. Her inference procedure is able to experiment with the unknown automaton, and is given, in response to each incorrect conjecture of the automaton's identity, a counterexample, a string that is misclassified by the conjectured automaton. Her algorithm exactly identifies the unknown automaton in time polynomial in the automaton's size and the length of the longest counterexample.

As mentioned above, a serious limitation of Angluin's procedure is its critical dependence on a means of *resetting* the automaton to a fixed start state. Thus, the learner can never really "get lost" or lose track of its current state since it can always reset the machine to its start state. In this chapter, we extend Angluin's algorithm, demonstrating that an unknown automaton can be inferred even when the learner is not provided with a reset.

This chapter also includes an improved version of Angluin's algorithm in the case that a reset *is* available; this improved algorithm significantly reduces the number of experiments that must be performed by the learner.

The generality of our results allows us to handle any "directed-graph environment," such

as the one in Figure 1. This means that we can handle many special cases as well, such as undirected graphs, planar graphs, and environments with special spatial relations. However, our procedures do not take advantage of such special properties of these environments, some of which could probably be handled more effectively. For example, we have found that permutation automata are generally easier to handle than non-permutation automata.

Previously, Rivest and Schapire [73, 75, 79] introduced the "diversity-based" representation of finite automata, an egocentric and often quite compact representation. They also described an algorithm that was proved to be effective for permutation automata, even in the absence of a reset. Some general techniques for handling non-permutation automata were also discussed; although not provably effective, these seemed to work well in practice for a variety of simple environments.

In this chapter, we generalize these results, demonstrating probabilistic inference procedures which are provably effective for both permutation and non-permutation automata. More generally, we present new inference procedures for the usual global state representation, as well as for the diversity-based representation.

Like Angluin, we assume that the inference procedures have an unspecified source of counterexamples to incorrectly conjectured models of the automaton. This differs from Rivest and Schapire's previous work where the learning model incorporated no such source of counterexamples; as already mentioned, this limitation makes learning of finite automata infeasible in the general case. For a robot trying to infer the structure of its environment, a counterexample is discovered whenever the robot's current model makes an incorrect prediction. For the special class of permutation automata, we show that an artificial source of counterexamples is unnecessary.

Our algorithms use powerful new techniques based on the inference of *homing sequences*. Informally, a homing sequence is a sequence of inputs that, when fed to the machine, is guaranteed to "orient" the learner: the outputs produced for the homing sequence completely determine the state reached by the automaton at the end of the homing sequence. Every finite-state machine has a homing sequence. For each inference problem, we show how a homing sequence can be used to infer the unknown machine, and how a homing sequence can be inferred as part of the overall inference procedure.

In sum, the main results of this chapter are four-fold: We describe efficient algorithms for inference of general finite automata using both the state-based and the diversity-based representations; both of these algorithms require a means of experimenting with the automaton and a source of counterexamples. Then, for permutation automata, we give efficient algorithms for both representations that do not require an external source of counterexamples. The time of the diversity-based algorithm for permutation automata beats the best previous bound by roughly

a factor of $D^3/\log D$, where D is the size of the automaton using the diversity-based representation. In the other three cases, our procedures are the first provably effective polynomial-time algorithms.

5-2   Two representations of finite automata

5-2.1   The global state-space or standard representation

An *environment* or *finite-state automaton* $\mathcal{E}$ is a tuple $(Q, B, \delta, q_0, \gamma)$ where:

- Q is a finite nonempty set of *states*,

- B is a finite nonempty set of *input symbols* or *basic actions*,

- δ is the *next-state* or *transition function*, which maps $Q \times B$ into Q,

- q_0, a member of Q, is the *initial state*, and

- γ is the *output function*, which maps Q into $\{0, 1\}$.

This is the standard, or state-based, representation.

For example, the graph of Figure 1 depicts the global state representation of an automaton whose states are the vertices of the graph, whose transition function is given by the edges, and whose output function is given by the shading of the vertices.

We denote the set of all finitely long action sequences by $A = B^*$, and we extend the domain of the function $\delta(q, \cdot)$ to A in the usual way: $\delta(q, \lambda) = q$, and $\delta(q, ab) = \delta(\delta(q, a), b)$ for all $q \in Q, a \in A, b \in B$. Here, λ denotes the empty or null string. Thus, $\delta(q, a)$ denotes the state reached by executing sequence a from state q; for shorthand, we often write qa to denote this state.

We say that $\mathcal{E}$ is a *permutation automaton* if for every action b, the function $\delta(\cdot, b)$ is a permutation of Q.

We refer to the sequence of outputs produced by executing a sequence of actions $a = b_1 b_2 \ldots b_r$ from a state q as the *output of a at q*, denoted $q\langle a \rangle$:

$$q\langle a \rangle = \langle \gamma(q), \gamma(qb_1), \gamma(qb_1b_2), \ldots, \gamma(qb_1b_2\ldots b_r) \rangle.$$

For instance, if the robot in Figure 1 executes action $a = \mathbf{xy}$ from its current state $q = 3$, then it will observe the sequence of actions

$$q\langle a \rangle = 3\langle \mathbf{xy} \rangle = \square \; \boxtimes \; \square \, .$$

(Don't confuse $\gamma(qa)$ and $q\langle a \rangle$. The former is a single value, the output of the state reached by executing a from q; for instance, $\gamma(qa) = \square$ in the example above. In contrast, $q\langle a \rangle$ is a $(|a| + 1)$-tuple consisting of the sequence of outputs produced by executing a from state q.)

Finally, for $a \in A$, we denote by $Q\langle a \rangle$ the set of possible outputs on input a:

$$Q\langle a \rangle = \{q\langle a \rangle : q \in Q\}.$$

Clearly, $|Q\langle a \rangle| \leq |Q|$ for any a.

Action sequence a is said to *distinguish* two states q_1 and q_2 if $q_1\langle a \rangle \neq q_2\langle a \rangle$. For instance **xy** distinguishes states 3 and 4 of the environment of Figure 1, but not states 1 and 2. We assume that $\mathcal{E}$ is *reduced* in the sense that, for every pair of distinct states, there is some action sequence which distinguishes them.

5-2.2   The diversity-based representation

In this section, we describe the second of our representations. See Rivest and Schapire's papers [73, 75, 79] for further background and detail. The representation is based on the notion of *tests* and *test equivalence*.

A *test* is an action sequence. (This definition differs slightly from that given in previous papers where the automata considered had multiple outputs (or "sensations") at each state.) The *value* of a test t at state q is $\gamma(qt)$, the output of the state reached by executing t from q.

Two tests t_1 and t_2 are *equivalent*, wri ten $t_1 \equiv t_2$, if the tests have the same value at every state. For instance, in the environment of Figure 1, tests **yxx** and **xx** are equivalent, as are tests **yy** and λ.

It's easy to verify that "$\equiv$" defines an equivalence relation on the set of tests. We write $[t]$ to denote the *equivalence class* of t, the set of tests equivalent to t. The value of $[t]$ at q is well defined as $\gamma(qt)$. The *diversity* of the environment, $D(\mathcal{E})$, is the number of equivalence classes of the automaton: $D(\mathcal{E}) = |\{[t] : t \in A\}|$. It can be shown that $\lg(|Q|) \leq D(\mathcal{E}) \leq 2^{|Q|}$, so the diversity of a finite automaton is always finite [73, 79].

The equivalence classes can be viewed as *state variables* whose values entirely describe the state of the environment. This is true because two states are equal (in a reduced sense) if and only if every test has the same value in both states.

It is often convenient to arrange the equivalence classes in an *update graph* such as the one in Figure 2 for the environment of Figure 1. Each vertex in the graph is an equivalence class so the size of the graph is $D(\mathcal{E})$. An edge labeled $b \in B$ is directed from vertex $[t_1]$ to $[t_2]$ if and only if $t_1 \equiv bt_2$. Note that each vertex has exactly one in-going edge labeled with each of the basic actions. This is because if $t_1 \equiv t_2$ then $bt_1 \equiv bt_2$.
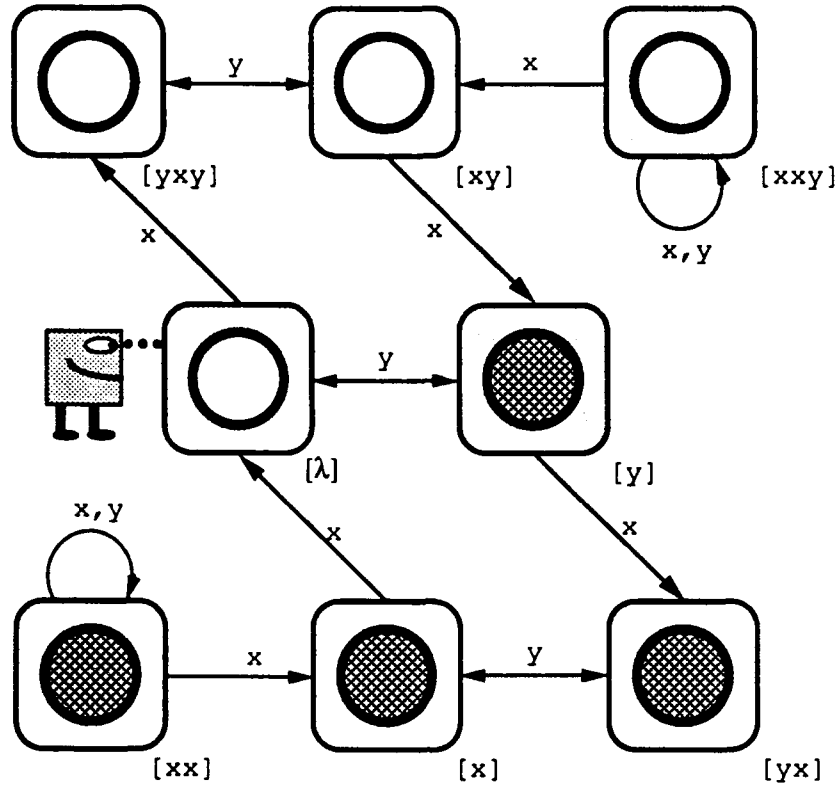
Figure 2: The update graph for the environment of Figure 1.

We associate with each vertex $[t]$ the value of t in the current state q. In the figure, we have used shading to indicate the value of each vertex in the robot's current state. The output of the current state is given by vertex $[\lambda]$, so this is the only vertex whose value can be observed by the robot. When an action b is executed from q, each vertex $[t]$ is replaced by the old value of $[bt]$, the vertex at the tail of $[t]$'s (unique) in-going b-edge. That is, in the new state qb, equivalence class $[t]$ takes on the old value of $[bt]$ in the starting state q. This follows from the fact that $\gamma((qb)t) = \gamma(q(bt))$. For instance, if action y is executed in the environment of Figures 1 and 2, then the value of $[\lambda]$ in the new state is ▨ , the old value of [y]; the new value of [yxy] is □ , the old value of [xy].

Thus, the value of each equivalence class in the state reached by executing any action can be determined easily using the update graph. Thus, the update graph can be used to simulate the environment.

Simple-assignment automata

The update graph can be viewed more abstractly as a special kind of automaton: A *simple-assignment automaton* S is a tuple $(V, B, \Upsilon, v_0, \omega)$ where:

- V is a finite nonempty set of *variables*,

- B is a finite nonempty set of *input symbols* or *basic actions*,

- Υ is the *update function*, which maps $V \times B$ into V,

- v_0, a member of V, is the *output variable*, and

- ω is the *initial-value function* which maps V into $\{0, 1\}$.

Here, we interpret V as a vector of state variables whose values determine the state of S. The initial values of these variables are given by ω, and the output of the machine is the current value of the special variable v_0. When an action $b \in B$ is executed, each variable v is updated in the new state with the old value of variable $\Upsilon(v, b)$. The function Υ can be extended to the domain $V \times A$ in the usual manner by defining $\Upsilon(v, \lambda) = v$ and $\Upsilon(v, ba) = \Upsilon(\Upsilon(v, a), b)$ for $v \in V$, $a \in A$ and $b \in B$. Thus, when a is executed, variable v is updated with the old value of variable $\Upsilon(v, a)$. In particular, this means that the output of S after executing $a \in A$ from the initial state is $\omega(\Upsilon(v_0, a))$.

Thus, the update graph is itself a simple-assignment automaton. In this case, the set V is the set of equivalence classes $\{[t] : t \in A\}$; the update function is defined by the rule $\Upsilon([t], b) = [bt]$; the output variable is $v_0 = [\lambda]$; and $\omega([t])$ is the value of $[t]$ in the initial state, $\gamma(q_0 t)$. With these definitions, it is straightforward to verify then that $\Upsilon(v_0, t) = [t]$, and so $\omega(\Upsilon(v_0, t)) = \gamma(\delta(q_0, t))$ for all $t \in A$.

On first blush, the structures of simple-assignment automata (such as the update graph of Figure 2) and of ordinary finite-state automata (such as the one given by the transition diagram of Figure 1) appear to be quite similar. In fact, their interpretations are very different. In the global-state representation, the robot moves from state to state while the output values of the states remain unchanged. On the other hand, in the diversity-based (or simple-assignment) representation, the robot remains stationary, only observing the output of a single variable ($[\lambda]$), and causing with its actions the values of the variables to move around. Thus, the diversity-based representation is more egocentric — the world is represented *relative* to the robot. In contrast, in the state-based representation, the world is represented by its *global* structure.

5-3   Homing sequences

Henceforth, we set $D = D(\mathcal{E})$, $n = |Q|$, $k = |B|$.

Input: $\mathcal{E}$ - a finite-state automaton
Output: h - a homing sequence
Procedure:
1 $h \leftarrow \lambda$
2 **while** $q_1\langle h \rangle = q_2\langle h \rangle$ but $q_1 h \neq q_2 h$ for some $q_1, q_2 \in Q$ **do**
3   let $x \in A$ distinguish $q_1 h$ and $q_2 h$
4      $h \leftarrow hx$
5 **end**

Figure 3: A state-based algorithm for constructing a homing sequence.

A *homing sequence* is an action sequence h for which the state reached by executing h is uniquely determined by the output produced: thus, h is a homing sequence if and only if

$$(\forall q_1 \in Q)(\forall q_2 \in Q)\ q_1\langle h \rangle = q_2\langle h \rangle \Rightarrow q_1 h = q_2 h.$$

For example, the string consisting of the single action "x" is a homing sequence for the environment of Figure 1. If $q\langle \mathbf{x} \rangle = \square\ \square$, then $q\mathbf{x} = 3$; if $q\langle \mathbf{x} \rangle = \square\ \boxtimes$, then $q\mathbf{x} = 2$; and, if $q\langle \mathbf{x} \rangle = \boxtimes\ \boxtimes$ then $q\mathbf{x} = 1$.

Kohavi [55] gives a complete discussion of homing sequences. He distinguishes between *preset* and *adaptive* homing sequences. Initially, we make use only of the former because they are simpler; later, we show that our inference procedures can be improved using adaptive homing sequences.

Given full knowledge of the structure of $\mathcal{E}$, it is easy to construct a homing sequence h, as shown in Figure 3. Initially, $h = \lambda$. On each iteration of the loop, a new extension x is appended to the end of h so that h now distinguishes two states not previously distinguished. Thus, $|Q\langle h \rangle| < |Q\langle hx \rangle| \leq n$, and therefore the program will terminate after at most $n - 1$ iterations. Further, since each extension need only have length $n - 1$ (see, for instance, Kohavi [55], Theorem 10-2), we have shown how to construct a homing sequence of length at most $(n-1)^2$.

A *diversity-based homing sequence* is an action sequence h which has the property that for every test t, there exists a prefix p of h such that $p \equiv ht$. For instance, it can be shown that $h = \mathbf{xxyx}$ is a diversity-based homing sequence for the environment represented in Figures 1 and 2. For example, if $t = \mathbf{yxy}$ then $ht = \mathbf{xxyxyxy} \equiv \mathbf{xxy}$.

Every diversity-based homing sequence h is a homing sequence. For if $q_1 h \neq q_2 h$ then there is some t for which $\gamma(q_1 ht) \neq \gamma(q_2 ht)$. Since ht is equivalent to some prefix p of h, we have $\gamma(q_1 p) \neq \gamma(q_2 p)$. Thus, $q_1\langle h \rangle \neq q_2\langle h \rangle$.

Figure 4 shows an algorithm for constructing a diversity-based homing sequence h. Again, h is built up from λ by appending extensions x. On each iteration, the cardinality of the set

Input: $\mathcal{E}$ - a finite-state automaton
Output: h - a diversity-based homing sequence
Procedure:
1 $h \leftarrow \lambda$
2 **while** $(\exists x \in A)(\forall p$ prefix of $h)$ $p \not\equiv hx$ **do**
3      $h \leftarrow hx$
4 **end**

Figure 4: A diversity-based algorithm for constructing a homing sequence.

$\{[p] : p$ prefix of $h\}$ increases by at least one; since the cardinality of this set is clearly bounded by D, there can be at most $D - 1$ iterations. Also, each extension need be no longer than $D - 1$. (For if $|x| \geq D$, then x has at least $D + 1$ suffixes, at least two of which must be equivalent. Thus, for some p, r, s, $x = prs$, $rs \equiv s$ and $r \neq \lambda$; therefore, ps is a shorter extension of h than x for which hps is inequivalent to every prefix of h.) Thus, we can find a diversity-based homing sequence of length at most $(D - 1)^2$.

Some other remarks about the length of homing sequences: First, the homing sequences constructed by the preceding algorithms are the best possible in the sense that there exist environments whose shortest homing sequence has length $\Omega(n^2)$ (or $\Omega(D^2)$). However, given a state-based (or a diversity-based) description of a finite-state machine, it is NP-complete to find the shortest homing sequence for the automaton. (This can be shown, for instance, by a reduction from exact 3-set cover.)

5-4 A state-based algorithm for general automata

In this and the next sections, we describe general algorithms for inferring the structure of an unknown environment $\mathcal{E}$.

We say that the learner has a *perfect model* of its environment if it can predict perfectly the output of the environment given any sequence of actions. The goal of our inference procedures is to construct a perfect model.

We assume that the learner is given access to $\mathcal{E}$, that the learner can observe the output of the environment when actions of its choosing are executed. We also assume that there is a "teacher" who provides the learner with counterexamples to incorrectly conjectured models of the environment. A counterexample is a sequence of actions whose true output from the current state differs from that predicted by the learner's model. Typically, there will be many sequences of actions which are counterexamples to a given conjecture, and by choosing an especially long or short counterexample, the teacher can significantly affect the running time of the procedure.

This fact is reflected in our running times which depend on the length of the counterexamples provided.

In the framework of a robot learning about its environment, we might imagine the robot, upon completion of a model of the environment which it believes to be correct, using that model to make predictions of the output of the environment's next state until an incorrect prediction is made. In this situation, the sequence of actions leading up to the error is the needed counterexample.

We generally assume that the unknown automaton is *strongly connected*, that is, every state can be reached from every other state:

$$(\forall q_1 \in Q)(\forall q_2 \in Q)(\exists a \in A)(q_1 a = q_2).$$

We make this assumption with little loss of generality: if $\mathcal{E}$ is not strongly connected, then an experimenting inference procedure, having no reset operation, will sooner or later fall into a strongly connected component of the state space from which it cannot escape, and so will have to be content thereafter learning only about that component.

This section focuses on an algorithm based on the global state representation for inferring an arbitrary unknown automaton.

5-4.1   Angluin's L^* algorithm

Our procedure is based closely on Angluin's L^* algorithm for learning regular sets [6]. Angluin shows how to efficiently infer the structure of any finite-state machine in the presence of what she calls a *minimally adequate teacher*. Such a teacher can answer two kinds of queries: On a *membership query*, the learner asks whether a given input string w is in the unknown language U, that is, whether the string is accepted by the unknown machine. On an *equivalence query*, the learner conjectures that the unknown machine is isomorphic to one it has constructed. The teacher replies that the conjecture is either correct or incorrect, and in the latter case provides a counterexample w, a string accepted by one machine but not the other.

The idea of Angluin's algorithm is to maintain an *observation table* (S, E, T). Here, S is a prefix-closed set of strings, and E is a suffix-closed set of strings. We can think of S as a set of strings that lead from the start state to the states of the automaton, and E as experiments which are executed from these states. The last variable T is a two-dimensional table whose rows are given by $S \cup SB$, and whose columns are given by E. Each entry $T(se)$, where $s \in S \cup SB$ and $e \in E$, records whether the string se is in the unknown language. For fixed s, Angluin denotes by $row(s)$ the vector of entries $T(se)$ for varying $e \in E$. Her algorithm extends S and E based on the results of queries, and ultimately outputs the correct automaton based on an equivalence between the states of the unknown machine and the distinct rows of the table

T. We denote by N_M and N_E the number of membership and equivalence queries made by L^*. These variables are implicit functions of n, k and m, where m is the length of the longest counterexample received. For Angluin's procedure L^*, we have $N_M = O(kmn^2)$, $N_E = n - 1$. However, in Section 5-4.5 below, we show how N_M can be improved to $O(kn^2 + n \log m)$.

In our framework, the learner could easily simulate Angluin's algorithm L^* if it were given a reset: to perform a membership query on w, the learner resets the environment, and executes the actions of w, observing the output of the last state reached. To perform an equivalence query on $\mathcal{E}'$, the learner resets the automaton and conjectures that $\mathcal{E}'$ is a perfect model of the environment. The teacher returns an action sequence w on which the conjectured model fails; this is the counterexample needed by L^*.

5-4.2 Using a homing sequence in lieu of a reset

However, in our model the learner is not provided with a reset. The main idea of our algorithm is to replace the reset with a homing sequence. In many respects, a homing sequence behaves like a reset: by executing the homing sequence, the learner discovers "where it is," what state it is at in the environment. However, unlike a reset, the final state is not fixed, and the learner does not know beforehand what state it will end up in. (Note that an automaton need not possess a *synchronizing sequence*, a sequence that forces the automaton into a given state independent of its starting state. So we use homing sequences instead.)

We begin by supposing that the learner has been provided with a correct homing sequence h. Later, we will show how to remove this assumption.

Suppose we execute h from the current state q, producing output $\sigma = q\langle h \rangle$. If we ever repeat this experiment from state q' and find $q'\langle h \rangle = \sigma$, then, because h is a homing sequence, the states where we finished must have been the same in both cases: $qh = q'h$. If we could guarantee that the output of h would continue to come up σ with good regularity, then we could simply infer $\mathcal{E}$ by simulating Angluin's algorithm, treating qh as the initial state. When L^* demands a reset, we execute h: if the output comes up σ, then we must be at qh, and our "reset" has succeeded; otherwise, try again. Unfortunately, in the general case, it may be very difficult to make h produce σ regularly.

Instead, we simulate an independent copy L_σ^* of L^* for each possible output σ of executing h, as shown in Figure 5. Since $|Q\langle h \rangle| \leq n$, no more than n copies of L^* will be created and simulated. Furthermore, on each iteration of the loop, at least one copy makes one query and so makes progress towards inference of $\mathcal{E}$. Thus, this algorithm will succeed in inferring $\mathcal{E}$ after no more than $n(N_M + N_E)$ iterations.

Input: access to $\mathcal{E}$, a finite-state automaton

h - a homing sequence for $\mathcal{E}$

Output: a perfect model of $\mathcal{E}$

Procedure:

```
 1 repeat
 2      execute h, producing output σ
 3      if it does not already exist, create L*_σ, a new copy of L*
 4      simulate the next query of L*_σ:
 5          if L*_σ queries the membership of action sequence a then
 6              execute a and supply L*_σ with the output of the final state reached
 7          if L*_σ makes an equivalence query then
 8              if the conjectured model E' is correct then
 9                  stop and output E'
10              else
11                  obtain a counterexample and supply it to L*_σ
12 end
```

Figure 5: A state-based algorithm for inferring $\mathcal{E}$ given a correct homing sequence.

5-4.3 Constructing a homing sequence

We now describe how to combine construction of the homing sequence h with the inference of $\mathcal{E}$. We maintain throughout the algorithm a sequence h which we presume is a true homing sequence. When evidence arises indicating that this is not the case, we will see how h can be extc ded and improved, eventually leading to the construction of a correct homing sequence. Initially, we take $h = \lambda$.

We use our presumably correct homing s\ ence h as described above and in Figure 5. If h is indeed a true homing sequence, we will of course succeed in inferring $\mathcal{E}$.

On the other hand, if h is incorrect, we may discover *inconsistent behavior* in the course of simulating some copy of L^*: suppose on two different iterations of the loop in Figure 5, we begin in states q_1 and q_2, execute h, produce output $q_1\langle h \rangle = q_2\langle h \rangle = \sigma$, and, as part of the simulation of L^*_σ, execute action sequence x. If h were a homing sequence, then x's output would have to be the same on both iterations since $q_1 h$ and $q_2 h$ must be equal.

However, if h is not a homing sequence, then it may happen that $q_1 h\langle x \rangle \neq q_2 h\langle x \rangle$. That is, we have discovered that x distinguishes $q_1 h$ and $q_2 h$, and so, just as was done in the algorithm of Figure 3, we replace h with hx, producing in a sense a "better" approximation to a homing sequence. At this point, the existing copies of L^* are discarded, and the algorithm begins from scratch (except for resetting h, of course). Since h can only be extended in this fashion $n - 1$ times, this only means a slowdown by at most a factor of n, compared to the algorithm of Figure 5.

Input: access to $\mathcal{E}$, a finite-state automaton
    n - the number of states of $\mathcal{E}$
Output: a perfect model of $\mathcal{E}$
Procedure:
1 $h \leftarrow \lambda$
2 **repeat**
3 execute h, producing output σ
4 if it does not already exist, create L_σ^*, a new copy of L^*
5 **if** $|\{row(s) : s \in S_\sigma\}| \leq n$ **then**
6   simulate the next query of L_σ^* as in Figure 5 (and check for inconsistency)
7 **else**
8   let $\{s_1, \ldots, s_{n+1}\} \subset S_\sigma$ be such that $row(s_i) \neq row(s_j)$
9   randomly choose a pair s_i, s_j from this set
10   let $e \in E_\sigma$ be such that $T_\sigma(s_i e) \neq T_\sigma(s_j e)$
11   with equal probability, re-execute either $s_i e$ or $s_j e$ (and check for inconsistency)
12 **if** inconsistency found executing some string x **then**
13   discard all existing copies of L^*
14   $h \leftarrow hx$
15 **until** a correct conjecture is made

Figure 6: A state-based algorithm for inferring $\mathcal{E}$.

Figure 6 shows how we have implemented these ideas. Here we have assumed n, the number of global states, has been provided to the learner. In fact, this assumption is entirely unnecessary. Although we omit the details, we can show that the stated bounds below hold (up to a constant) for a slightly modified algorithm which does not require that the learner be explicitly provided with the value of n. The trick is the usual one of repeatedly doubling our estimate of n.

Recall that L^* requires maintenance of an observation table (S, E, T). Let $(S_\sigma, E_\sigma, T_\sigma)$ denote the observation table of L_σ^*. Of course, T_σ can only record output produced when executing an action sequence from what is only *presumed* to be a fixed initial state.

Angluin's analysis implies that if L_σ^* makes more than $N_M + N_E$ queries, then the number of distinct rows will exceed n. This can only happen if h is not a homing sequence, but how do we know how to correctly extend h if we have not actually seen an inconsistency? We show that if an inconsistency has not been found by the time the number of rows exceeds n, then we can use a probabilistic strategy to find one quickly with high probability.

Suppose we execute h from state q, with output σ, and we find that for L_σ^*, there are more than n distinct rows. Then, as in Figure 6, there exist strings $s_1, \ldots, s_{n+1}$ in S_σ whose rows are all distinct. By the pigeon-hole principle, there is at least one pair of distinct rows s_i, s_j such that $qhs_i = qhs_j$. Further, since $row(s_i) \neq row(s_j)$, there is some $e \in E_\sigma$ for which

$T_\sigma(s_i e) \neq T_\sigma(s_j e)$. However, $\gamma(qhs_i e) = \gamma(qhs_j e)$. Therefore, either $\gamma(qhs_i e) \neq T_\sigma(s_i e)$ or $\gamma(qhs_j e) \neq T_\sigma(s_j e)$, and so re-executing $s_i e$ (or $s_j e$, respectively) from the current state qh will produce the desired inconsistency. (Recall that T_σ records the results of previous executions of these strings.)

So the chance of randomly choosing the correct pair s_i, s_j as above is at least $\binom{n+1}{2}^{-1}$, and the chance of then choosing the correct experiment to re-run of $s_i e$ or $s_j e$ is at least $1/2$. Thus, the probability of finding an inconsistency using the technique of Figure 6 in this situation is at least $1/n(n+1)$. Repeating this technique $n(n+1)\ln(1/\delta)$ times gives a probability of at least $1 - \delta$ of finding an inconsistency. Also, no more than n^2 copies of L^* are ever created, and $|h|$ does not exceed $O(n^2 + nm)$ since h is extended at most $n - 1$ times, and each extension has length $O(n + m)$. a bound on the length of any query required by L^*.

Putting these facts together, we have proved:

Theorem 4.1 *Given $\delta > 0$, the algorithm described in Figure 6 halts and outputs a perfect model with probability at least $1 - \delta$ in time polynomial in n, m, k and $1/\delta$, and after executing*

$$O(n^3(n + m)(n^2 \log(n/\delta) + N_M + N_E))$$

actions.

If we assume $m = O(n)$ and $k = O(1)$ and use the previously given bounds on N_M and N_E, then the number of actions executed by the procedure (and the running time as well) simplifies to $O(n^6 \log(n/\delta))$.

The procedure can be modified, replacing the preset homing sequence which we have been using with an adaptive one whose input at each step depends on the output seen up to that point. This modification shaves a factor of n off the bound given above, and is described in greater detail in the next section.

It is an open question whether this bound can be significantly tightened. It seems likely that an algorithm which combines the many copies of L^* into one would have a superior running time, but we have not been successful in implementing this intuition.

5-4.4 Adaptive homing sequences

The algorithm of Figure 6 is certainly quite wasteful in that, when h is discovered not to be a homing sequence, everything is thrown away and the algorithm starts over from scratch. As a result, up to n copies of L^* are discarded each time h is extended. Since h can be extended up to $n - 1$ times. this means as many as n^2 copies of L^* may eventually be simulated by the algorithm.
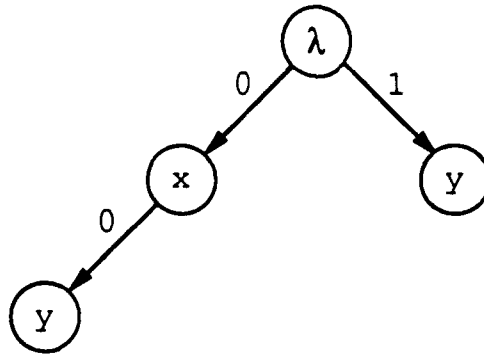
Figure 7: An example adaptive homing sequence.

In this section. we describe a way of modifying the procedure so that only one copy of L^* is discarded when h is extended, leading to an $O(n)$ bound on the total number of copies of L^* simulated.

As mentioned above, the idea of the modification is to replace our preset homing sequence (the kind described up to this point) with an adaptive one. In many ways, preset homing sequences are rather inefficient tools. For example, it may be that, starting from some states, executing only half the sequence is sufficient to reach a state uniquely determined by the observed output. An adaptive homing sequence is a much more intelligent kind of homing sequence. It is like a preset homing sequence in that the output observed can be used to determine the state reached. However. the difference is that the action executed at each step may depend on the output observed up to that point.

Despite its name. an *adaptive sequence* a is not a sequence at all but a decision tree with the following properties: The root node of a is labeled λ. and each of the other nodes in the tree is labeled with one of the basic actions in B. Every node has at most one 0-child. and at most one 1-child. An example adaptive sequence is given in Figure 7.

An adaptive sequence is executed in a natural manner: We begin at the root node. If the output of the current state is 0 (or 1) then we proceed to the 0-child (1-child) of the root. The basic action labeling the node reached is then executed, and, based on the resulting output, we proceed down the tree in the same fashion, at each step branching to the 0- or 1-child depending on the output observed. This continues until we "fall off" the tree, i.e., until it is necessary to move to a node that does not exist in the tree.

For example. if the tree of Figure 7 is executed from the current state of Figure 1. then the action sequence "x" will be executed producing output ☐ ▨ ; if the sequence is executed from state 4. then "xy" will be executed with output ☐ ☐ ▨ . (An adaptive homing sequence can

be naturally defined in terms of any set of output symbols, such as ☐ and ▓ , rather than 0 and 1.)

As with ordinary sequences, we write qa to denote the state reached by executing adaptive sequence a from state q, and we write $q\langle a \rangle$ to denote the output produced by executing a from state q. Thus, as for preset homing sequences, an *adaptive homing sequence* is an adaptive sequence h for which $q_1\langle h \rangle = q_2\langle h \rangle$ only if $q_1 h = q_2 h$ for all $q_1, q_2 \in Q$.

Modifying the algorithm

We are now ready to describe how the algorithm of Figure 6 can be modified to use adaptive rather than preset homing sequences. The structure of the algorithm is not changed at all. Nor is the simulation of queries, the handling of over-sized copies of L^*, etc. Only the construction of the adaptive homing sequence h is modified.

Initially, h is chosen to be the adaptive sequence consisting just of a root node λ. As before, h is repeatedly executed; each time, its output selects a copy of L^*. A query of the selected copy is then simulated, just as before. Now, however, a detected inconsistency is handled differently: Suppose an inconsistency is found executing x. More precisely, suppose that on two different iterations, we began in states q_1 and q_2, executed h, and observed output $q_1\langle h \rangle = q_2\langle h \rangle = \sigma$. Further, when x was then executed, it was discovered that $q_1 h\langle x \rangle \neq q_2 h\langle x \rangle$. This latter fact implies that $q_1 h \neq q_2 h$, and so h cannot be an adaptive homing sequence. As before, we would like to use x to repair h: we would like to "graft" x onto tree h so that the resulting tree h' distinguishes q_1 and q_2.

In fact, this can be done quite easily: Let v_0 be the last node of h visited when h is executed from q_1 and q_2 (it must be the same node in both cases since $q_1\langle h \rangle = q_2\langle h \rangle$), and let $x = b_1 \cdots b_r$, where $b_i \in B$. Note that v_0 has no $\gamma(q_1 h)$-child since this marks the "fall-off" point of h. The grafted tree h' is the same as h accept that in h', node v_0 has a $\gamma(q_1 h)$-child which is the root of a linear subtree corresponding to the execution of x. More precisely, in h', each node v_{i-1} has a $\gamma(q_1 h b_1 \cdots b_{i-1})$-child v_i labeled b_i, for $1 \leq i \leq r$.

It can be verified that $q_1\langle h' \rangle \neq q_2\langle h' \rangle$. Thus $|Q\langle h \rangle| < |Q\langle h' \rangle| \leq n$, and so h will be grafted in this fashion at most $n - 1$ times.

So line 13 in Figure 6 is replaced by a call to a grafting subroutine as described above. Further, since h and h' are the same except for node v_0, it is no longer necessary to discard all copies of L^* — it is sufficient to discard only L^*_σ, the copy on which an inconsistency was discovered. Thus, since $|Q\langle h \rangle|$ increases each time a single copy of L^* is discarded, at most $n - 1$ copies are ever discarded throughout the execution of the algorithm. Since the number of copies in existence at any one time is also bounded by n, it follows that at most $2n - 1$ copies of L^* are simulated by this modified procedure. Thus, this improves the bound given in Theorem 4.1 by a factor of $\theta(n)$.

5-4.5   Improving Angluin's L^* algorithm

In this section, we describe a variant of Angluin's L^* algorithm that significantly improves the worst-case number of membership queries made by the inference procedure. This, in turn, leads to immediate improvements in the performance of our homing sequence algorithms.

As mentioned above, Angluin's algorithm maintains an observation table (S, E, T). The function or table T records the value $T(x) = \gamma(q_0 x)$ for each string $x \in (S \cup SB)E$. (Here, q_0 is $\mathcal{E}$'s initial state to which, in Angluin's model, the automaton can always be reset.) The entries of T are filled in using membership queries, and it follows that the number of queries needed is just the cardinality of $(S \cup SB)E$. For Angluin's algorithm, $|S|$ is bounded by $O(mn)$, and $|E|$ by $O(n)$. Our algorithm improves on Anlguin's by limiting $|S|$ to just n; however, to achieve this bound on $|S|$, $n \lg m$ additional queries will be needed, giving an overall bound of $O(kn^2 + n \log m)$ on the required number of membership queries.

As mentioned earlier, S is a prefix-closed set of strings representing states of $\mathcal{E}$. Unlike Angluin's algorithm, ours maintains the condition that $q_0 s_1 \neq q_0 s_2$ if $s_1 \neq s_2$ for all $s_1, s_2 \in S$. Thus, $|S| \leq n$ at all times. Also, S only grows in size (strings are never deleted from S). The set E represents a set of experiments which distinguish the states of S (i.e., the states $q_0 s$ for $s \in S$).

Here is an outline of our algorithm, which is very similar to Angluin's. Initially, S and E are initialized to the set $\{\lambda\}$. Using membership queries, fill in the entries of table T, and make (S, E, T) closed (discussed below). Then, from (S, E, T), construct and conjecture machine $\mathcal{E}'$. If the conjecture is correct, quit. Otherwise, update the set E using the returned counterexample, and repeat until a correct conjecture is made.

We say observation table (S, E, T) is *closed* if for all $s \in SB$ there exists $s' \in S$ such that $row(s) = row(s')$. (Recall that $row(s)$ is that function $f : E \rightarrow \{0, 1\}$ for which $f(e) = T(se)$.) If $s \in SB$ witnesses that $(S, E. T)$ is not closed, then s is simply added to S (and T updated using membership queries). Note that this maintains the condition that all rows of S are distinct (and thus, the states to which they lead are also distinct).

(Angluin's algorithm also requires that the observation table be *consistent*, that is, that $row(s_1 b) = row(s_2 b)$ whenever $row(s_1) = row(s_2)$ for $s_1, s_2 \in S$ and $b \in B$. However, since our algorithm maintains the condition that $row(s_1) \neq row(s_2)$ for $s_1 \neq s_2$, this condition is always trivially satisfied.)

The conjectured machine $\mathcal{E}' = (Q', B, \delta', q_0', \gamma')$ is constructed in a natural manner: its state set is $Q' = S$ with initial state $q_0' = \lambda$; its output function is defined by $\gamma'(s) = T(s)$; and its transition function is given by $\delta'(s, b) = s'$ where s' is that unique member of S for which $row(sb) = row(s')$.

Finally, if $\mathcal{E}'$ is different from $\mathcal{E}$, a counterexample z is obtained, and the set E must be

updated. Our algorithm adds only a single string to E using z. However, to find this string, the procedure makes up to $\lg |z|$ membership queries.

The key property that must be satisfied by the new experiment e (which will be added to E) is the following: for some $s, s' \in S$ and $b \in B$ for which $row(s) = row(s'b)$, it must be that $\gamma(q_0se) \neq \gamma(q_0s'be)$. That is, experiment e must witness that q_0s and $q_0s'b$ are different states. If this property is satisfied, then adding e to E will cause $|S|$ to increase by at least one (to maintain closure) so that the total number of equivalence queries is bounded by $n - 1$.

We now describe how such an experiment can be found. For $0 \leq i \leq |z|$, let p_i, r_i be such that $z = p_i r_i$, and $|p_i| = i$. Let $s_i = \delta'(\lambda, p_i)$ be the state reached in $\mathcal{E}'$ after the first i symbols of z have been executed. Then on input z, machine $\mathcal{E}$ reaches a state outputting the value $\gamma(q_0 z) = \gamma(q_0 s_0 r_0)$. (Assume this value is 0.) On the other hand, on input z, machine $\mathcal{E}'$ reaches a state outputting the value $\gamma'(\delta'(\lambda, z)) = \gamma'(s_{|z|}) = T(s_{|z|}) = \gamma(q_0 s_{|z|} r_{|z|})$. Since z is a counterexample, this value must be 1.

Let $\alpha_i = \gamma(q_0 s_i r_i)$. Note that, by simulating $\mathcal{E}'$, s_i can be computed, and so α_i can be determined with a membership query for any i. From the comments above, we have that $\alpha_0 = 0$ and $\alpha_{|z|} = 1$. Using a kind of binary search, we can find some i such that $\alpha_i \neq \alpha_{i+1}$ (such an i clearly must exist): first we query $\alpha_{|z|/2}$; if the result is 1, then query $\alpha_{|z|/4}$; otherwise, query $\alpha_{3|z|/4}$, etc. In this manner, such an i can be found in $\lg |z|$ queries.

We claim then that r_{i+1} is the desired experiment: Let b be the first symbol of r_i. Then $\gamma(q_0 s_i b r_{i+1}) = \gamma(q_0 s_i r_i) = \alpha_i \neq \alpha_{i+1} = \gamma(q_0 s_{i+1} r_{i+1})$. However, by definition of s_i, we have $s_{i+1} = \delta'(s_i, b)$ and so $row(s_{i+1}) = row(s_i b)$. Thus, as argued above, adding r_{i+1} to E causes $|S|$ to increase. It follows that at most $n - 1$ equivalence queries are required by the algorithm. For each equivalence query, $\lg m$ membership queries are needed to find the right experiment to add to E. Also, since $|E| \leq n$ and $|S| \leq n$, at most $|(S \cup SB)E| \leq (k + 1)n^2$ membership queries are needed to record the entries of T. Finally, it can be seen that each membership query has length at most $n + m$. The procedure is clearly polynomial time, and its correctness follows from arguments given above and by Angluin.

In a quite naive implementation of the algorithm, the rows of S are filled in first, and, once a row of some string in SB has been completed, it is compared in $O(n^2)$ time to every other row of S until an identical row is discovered, or until it is determined that there is no other identical row in S (in which case, (S, E, T) is not closed). For even such a naive implementation, it can be verified that each query requires processing time that is at worst proportional to the bound of $O(n^2 + nm)$ on the length of h in the algorithm of Figure 6.

Combining this improvement to L^* with the adaptive homing sequence ideas described in Section 5-4.4, we thus have shown:

Theorem 4.2 *There exists an algorithm that halts and outputs a perfect model of any finite-*

state environment $\mathcal{E}$ with probability at least $1 - \delta$. The algorithm's running time, and the number of actions executed are both bounded by

$$O(n^3(n + m)(n\log(n/\delta) + kn + \log m)).$$

5-5 A diversity-based algorithm for general automata

In this section, we describe a diversity-based algorithm for inferring finite automata in the general case. The idea of the algorithm is to construct a simple-assignment automaton that is equivalent to the update graph.

Our algorithm maintains a suffix-closed set T of tests which will act as the variables of the constructed simple-assignment automaton. A test t is added to T only after it has been determined that t is inequivalent to every test already in T. Thus, at all times, $|T| \leq D$, and each test of T represents a different test-equivalence class (i.e., a node of the update graph); naturally, we would like eventually that all of the equivalence classes be represented.

Additionally, a function or table $r : BT \to 2^T$ is maintained with the interpretation that $r(x)$ represents those tests in T which are plausibly equivalent to x. That is, initially $r(x) = T$, and a test $t \in T$ is removed from $r(x)$ when it has been determined that $t \not\equiv x$.

Note that if $|T| = D$ (so that every equivalence class is represented in T), and if, for all $x \in BT$, $r(x)$ is a singleton $\{s_x\}$ for some $s_x \in T$, then $s_x \equiv x$ and a simple-assignment automaton isomorphic to the update graph is easily constructed: its variable set is T, its output variable is λ, and its update function Υ is defined by $\Upsilon(t,b) = s_{bt}$. (The initial values function ω is handled below.)

The simple-assignment automata conjectured by our algorithm are constructed from T and r in a very similar manner. We choose $V = T$ and $v_0 = \lambda$. However, in general, it may not be the case that $|r(x)| = 1$ for all $x \in BT$. Therefore, we choose $\Upsilon(t,b)$ to be an *arbitrary* element of $r(bt)$. If one or more of our choices is incorrect, then we can use the provided counterexample to correct our error. More precisely, we show that, using experiments and counterexamples to this conjectured automaton, we can find $t \in T$ and $b \in B$ such that $\Upsilon(t,b) \not\equiv bt$. That is, our choice for $\Upsilon(t,b)$ can be removed from $r(bt)$. Thus, for some $x \in BT$, $r(x)$ is reduced in size.

Also, note that if $r(x)$ is reduced to the empty set, then x is inequivalent to every member of T, and so can itself be added to T. The table r is then updated appropriately. Since $|T| \leq D$, since $r(x) \subset T$, and since some $r(x)$ shrinks on each iteration, it follows that this simplified algorithm converges to a perfect model after at most $(k + 1)D^2$ iterations.

5-5.1   An algorithm that uses a provided homing sequence

As in Section 5-4, we assume initially that a diversity-based homing sequence h is given. Later, we show how h can be constructed.

Let t be any test. Then ht is equivalent to some prefix of h. For selected tests t in A, we maintain *candidate sets* $C(t) \subset \{0, \ldots, |h|\}$ representing the prefixes of h which are plausibly equivalent to t. Let h_i denote that prefix of h of length i. Initially, $C(t) = \{0, \ldots, |h|\}$, and, when it has been determined that $h_i \not\equiv ht$, index i is removed from the set. Note that when ht is executed from some state q, both of the outputs $\gamma(qh_i)$ and $\gamma(qht)$ are observed since h_i is a prefix of ht. Thus, if we find these outputs differ, then clearly $h_i \not\equiv ht$ and so i can be deleted from $C(t)$.

Suppose h has been executed from some state q producing output $\sigma = \langle \sigma_0, \ldots, \sigma_{|h|} \rangle$. We say that a set $X \subset \{0, \ldots, |h|\}$ is *coherent* (with respect to σ) if $\sigma_i = \sigma_j$ for $i, j \in X$. If X is coherent, then the common value of all σ_i with $i \in X$ is called X's *selected value* (with respect to σ), and it is denoted $\sigma[X]$.

Note that, if $C(t)$ is coherent, then the value of t in the current state qh is known — it is just $C(t)$'s selected value. On the other hand, if candidate set $C(t)$ is incoherent, then if t is executed, at least one element of $C(t)$ will be eliminated.

What's more, if i is eliminated from $C(t)$, then every other index j for which $h_i \equiv h_j$ is also removed since the two tests have the same value in every state. That is, $|\{[h_i] : i \in C(t)\}|$ decreases by at least one. Thus, $C(t)$ can be reduced in this fashion at most $D - 1$ times.

Also, if we find for tests t_1 and t_2 that $C(t_1)$ and $C(t_2)$ are disjoint, then t_1 and t_2 cannot possibly belong to the same equivalence class. Moreover, if for any $a \in A$ we find that $C(at_1)$ and $C(at_2)$ are disjoint, then $at_1 \not\equiv at_2$ and therefore $t_1 \not\equiv t_2$. This is the primary technique used by our procedure for determining inequivalence of tests (and thus for the elimination of tests from $r(x)$).

Our algorithm maintains a candidate set for each $t \in T$. If all of these candidate sets are coherent (after h has been executed from some state q), then the value of every test $t \in T$ is known in the current state qh; these values are used then to determine the function ω in the conjectured automaton. Specifically, if all the candidate sets for the tests in T are coherent, then a conjecture may be made in which V, v_0 and Υ are as described above, and $\omega(t)$ is taken to be the selected value of $C(t)$ (which is, from the preceding remarks, the value of t in the current state).

We describe next how a counterexample z to such a conjecture S is handled. The technique is similar to that described in Section 5-4.5. Let $z = p_i s_i$ where $|p_i| = i$ for $0 \leq i \leq |z|$. Let $t_i = \Upsilon(\lambda, s_i)$. Finally, let $u_i = p_i t_i$. We maintain henceforth a candidate set for each test u_i. Our hope is that these candidate sets will be reduced to the point that, for some i,

$C(u_i) \cap C(u_{i+1}) = \emptyset$. For if this happens, then we can conclude that $u_i \not\equiv u_{i+1}$. Noting that $u_i = p_i t_i$ and $u_{i+1} = p_i b t_{i+1}$ where b is the last symbol of p_{i+1}, this implies that $t_i \not\equiv b t_{i+1}$. Since $t_i = \Upsilon(t_{i+1}, b) \in r(b t_{i+1})$, it follows that t_i can be deleted from $r(b t_{i+1})$ as desired.

We show that $C(u_0)$ and $C(u_{|z|})$ are disjoint. Thus, if the candidate sets $C(u_i)$ can be sufficiently reduced, eventually two consecutive sets $C(u_i)$ and $C(u_{i+1})$ will be made disjoint as needed. Note that the conjectured automaton S predicted that the value of z in the current state qh is $\omega(\Upsilon(\lambda, z)) = \omega(t_0)$. Assume this value is 0. Then, by ω's definition, $C(u_0) = C(t_0) \subset \sigma^{-1}(0)$, where $\sigma^{-1}(x) = \{0 \le i \le |h| : \sigma_i = x\}$. On the other hand, since z is a counterexample, $\gamma(qhz) = 1$. Thus, if z is executed from the current state, then $C(u_{|z|}) = C(z)$ will be included in $\sigma^{-1}(1)$, and, as claimed $C(u_0) \cap C(u_{|z|})$ will be empty.

Unfortunately, to continually reduce the sets $C(u_i)$, these sets must continually be found incoherent. This may be a problem because they may very well all be found to be coherent without any consecutive pair being disjoint. To handle this situation, our algorithm makes a new conjecture that leaves V, v_0 and Υ alone, but which chooses ω appropriately as described above. This gives a new sequence of tests u_i for which candidate sets must also be maintained. We show below that no more than $D - 1$ such sequences need ever be started by the algorithm before one of the sets $r(x)$ is reduced.

The complete algorithm is shown in Figure 8. In the figure, when a counterexample z is received, a sequence of tests $u_{\ell 0}, \ldots, u_{\ell m_\ell}$ is constructed as described above; variable ℓ counts the number of such counterexamples received for the same choice of Υ. The set $K(i, j)$ is a candidate set for test u_{ij}. Note that the same test may have several candidate sets, not necessarily the same: even if $u_{ij} = u_{i'j'}$, it may be that $K(i, j) \ne K(i', j')$ if $\langle i, j \rangle \ne \langle i', j' \rangle$. Although this may seem inefficient, it appears to be necessary for proving the algorithm's correctness.

Also, candidate sets are updated in the obvious way: if $t \in T$ is executed leading to a state outputting the value x, then $C(t) \leftarrow C(t) \cap \sigma^{-1}(x)$ (and similarly for sets $K(i, j)$). Note that only the specified candidate set is modified.

Theorem 5.1 *The algorithm described in Figure 8 halts in polynomial time after executing at most*

$$O(kmD^4(|h| + D + m))$$

actions, and outputs a perfect model.

Proof: If the algorithm halts, then it outputs a perfect model. Therefore, it suffices to prove that it halts having executed only the stated number of actions.

Most of the arguments needed to prove this theorem were given above. Here, we try to pull those arguments together, filling in missing details. Below, we say that a property holds *on each iteration* if it holds between each iteration of the main loop.

Input: access to $\mathcal{E}$, a finite-state automaton

   h - a diversity-based homing sequence for $\mathcal{E}$

Output: a perfect model of $\mathcal{E}$

Procedure:

1 $T \leftarrow \{\lambda\}; C(\lambda) \leftarrow \{0, \ldots, |h|\}$

2 $r(b) \leftarrow T$, $\Upsilon(\lambda, b) \leftarrow \lambda$ **for** $b \in B$

3 $\ell \leftarrow 0$

4 **repeat**

5   execute h, producing output σ

6   **if** $C(t)$ is incoherent for some $t \in T$ **then**

7   execute t and update $C(t)$

8   **else if** $K(i,j)$ is incoherent for some $1 \le i \le \ell$, $0 \le j \le m_i$ **then**

9   choose the smallest i for which $K(i,j)$ is incoherent for some $0 \le j \le m_i$

10   execute u_{ij} and update $K(i,j)$

11   **else**

12        $\omega(t) \leftarrow \sigma[C(t)]$ **for** $t \in T$

13   conjecture $\mathcal{S} = (T, B, \Upsilon, \lambda, \omega)$

14   **if** $\mathcal{S}$ is a perfect model **then**

15   stop and output $\mathcal{S}$

16   **else**

17   obtain counterexample z

18            $\ell \leftarrow \ell + 1$; $m_\ell \leftarrow |z|$

19   **for** $0 \le j \le m_\ell$:

20                $u_{\ell j} \leftarrow p_j \cdot \Upsilon(\lambda, s_j)$ where $z = p_j s_j$ and $|p_j| = j$

21                $K(\ell, j) \leftarrow \{0, \ldots, |h|\}$

22            $K(\ell, 0) \leftarrow \sigma^{-1}(\omega(u_{\ell 0}))$

23   execute $z = u_{\ell m_\ell}$ and update $K(\ell, m_\ell)$

24   **if** $K(i,j) \cap K(i, j+1) = \emptyset$ for some $1 \le i \le \ell$, $0 \le j < m_i$ **then**

25        $x \leftarrow b_0 t_0$ where $u_{i,j+1} = p b_0 t_0$, $|p| = j$ and $b_0 \in B$ [this implies $u_{ij} = p \cdot \Upsilon(t_0, b_0)$]

26        $r(x) \leftarrow r(x) - \{\Upsilon(t_0, b_0)\}$

27   **if** $r(x) = \emptyset$ **then**

28            $r(t) \leftarrow r(t) \cup \{x\}$ **for** $t \in BT - T$

29            $T \leftarrow T \cup \{x\}$; $C(x) \leftarrow \{0, \ldots, |h|\}$

30            $r(bx) \leftarrow T$ **for** $b \in B$

31        $\Upsilon(t, b) \leftarrow$ any member of $r(bt)$ **for** $b \in B, t \in T$

32        $\ell \leftarrow 0$

33 **end**

Figure 8: A diversity-based algorithm for inferring $\mathcal{E}$ given a diversity-based homing sequence.

First, on each iteration, if $i \notin C(t)$ then $h_i \not\equiv ht$ for any t. This follows from the manner in which C is updated. Also, the contrapositive implies that $C(t)$ is non-empty on each iteration since $h_i \equiv ht$ for some i since h is a diversity-based homing sequence. These statements hold also for candidate sets $K(i,j)$. (At line 22, this follows from the fact that, by ω's definition, $C(u_{\ell 0}) \subset \sigma^{-1}(\omega(u_{\ell 0}))$.)

These facts imply that, on each iteration, $t \notin r(x)$ only if $x \not\equiv t$, for $t \in T$, $x \in BT$. Note also that $r(x)$ is non-empty on each iteration.

Thus, if the last element of $r(x)$ is eliminated, then $x \not\equiv t$ for all $t \in T$, and so it follows that the tests in T are pairwise independent. Thus, by definition of diversity, $|T| \leq D$ on each iteration, and so lines 25–32 are executed at most $(k+1)D^2$ times; in particular, this implies that ℓ is reset to zero at most this many times.

We say a set x *respects* set y if either $x \subset y$ or $x \cap y = \emptyset$.

By definition of equivalence, and also because of the manner in which C is updated, the set $\{0 \leq i \leq |h| : h_i \equiv x\}$ respects $C(t)$ for any tests t and x, on each iteration. Thus, $C(t)$ can be reduced in size at most $D-1$ times (and similarly for $K(i,j)$). Combined with the fact that $|T| \leq D$, this implies that the condition at line 6 is satisfied at most $D(D-1)$ times.

We will show that $\ell \leq D-1$ on each iteration. This will complete the theorem: Since ℓ is reset to zero at most $(k+1)D^2$ times, the condition at line 8 can be satisfied at most $(k+1)mD^2(D-1)^2$ times. Also, since ℓ is incremented at line 18, the conditions at lines 6 and 8 can fail to be satisfied at most $(k+1)D^2(D-1)$ times. This gives us an overall bound on the total number of iterations of the main loop, and, since at most $|h| + m + D - 1$ actions are executed on each iteration, the result follows.

Thus, to complete the proof, we show that $\ell \leq D-1$ on each iteration.

First, note that on each iteration, $K(i,j) \cap K(i,j+1) \neq \emptyset$ for $1 \leq i \leq \ell$ and $0 \leq j < m_i$. Also, because the *smallest* i is chosen at line 9, the set $K(i,j)$ is reduced by the algorithm only when $K(i',j')$ is coherent for all $1 \leq i' < i$ and $0 \leq j' \leq m_{i'}$. Also, a counterexample is obtained only when every set $K(i,j)$ is coherent. Thus, it follows that on each iteration $K(i',j')$ respects $K(i,j)$ for $i' < i$. (When $K(i,j)$ is reduced, either all or none of the elements of $K(i',j')$ are deleted.)

To prove $\ell \leq D-1$, we define a sequence of undirected graphs $G_0, \ldots, G_\ell$. The vertex set of each graph is the set $\{0, \ldots, |h|\}$. In G_i, an edge connects two vertices r and s if and only if $h_r \equiv h_s$ or $\{r,s\} \subset K(i',j)$ for some $1 \leq i' \leq i$, $0 \leq j \leq m_{i'}$.

We will be interested in counting the number of connected components of each graph G_i. First, note that G_0 has at most D connected components by definition of diversity. We will show that each graph G_{i-1} has at least one more connected component than G_i. Since every (non-empty) graph has at least one connected component, this implies that $\ell \leq D-1$.

Since the edge set of G_{i-1} is a subset of the edge set of G_i, it suffices to find a single pair of vertices which are connected in G_i, but not in G_{i-1}.

As argued above in discussing the handling of counterexamples, the sets $K(i, 0)$ and $K(i, m_i)$ are disjoint. Let r and s be respective members of these sets. Then r and s are connected in G_i because, as remarked above, $K(i, j) \cap K(i, j + 1) \neq \emptyset$ on each iteration, for $0 \leq j < m_i$.

We claim that r and s are not connected in G_{i-1}. For if they were, then since r but not s is contained in $K(i, 0)$, there must be adjacent vertices r' and s' on the path from r to s for which r' but not s' is contained in $K(i, 0)$. Since r' and s' are adjacent, either $h_{r'} \equiv h_{s'}$ or $\{r', s'\} \subset K(i', j)$ for some $i' < i$. However, as already argued, either case implies that $\{r', s'\}$ respects $K(i, 0)$, a contradiction.

This completes the proof.   ∎

So Theorem 5.1 shows that an effective diversity-based algorithm exists for inferring a finite-state environment, assuming a diversity-based homing sequence has been provided. We turn next to the problem of extending this algorithm to handle environments when such a sequence is not available.

5-5.2   Constructing a homing sequence

As in the algorithm of Figure 6, we presume that some sequence h is a true diversity-based homing sequence until it becomes necessary to extend and improve h. Initially, $h = \lambda$. If for some test x, candidate set $C(x)$ is reduced to the empty set, then clearly h cannot be a diversity-based homing sequence since this implies that hx is inequivalent to every prefix of h. We therefore replace h with hx as is done in the algorithm of Figure 4. Since more equivalence classes are represented by the prefixes of hx than by those of h, it follows that h must converge to a correct homing sequence if extended in this fashion at most $D - 1$ times.

Our extended algorithm is quite similar to the one given in Figure 8. As before, we maintain a set T and function r, which together record inequivalences determined among the tests. Now, however, the problem of determining that two tests are inequivalent becomes more difficult: we saw earlier that if h is a diversity-based homing sequence and $C(x) \cap C(y) = \emptyset$ for two tests x and y, then $x \not\equiv y$. However, if h is *not* a diversity-based homing sequence, this conclusion may be false — it may be that x and y are equivalent to one another, but not to any prefix of h.

Nevertheless, we show that if x and y are in fact equivalent, then by re-running these tests repeatedly in an appropriate manner, we can with high probability eliminate all the elements of one of the candidate sets, thus yielding an extension to h as described above.

Suppose that, having executed h, we find that $C(x)$ and $C(y)$ are coherent, and furthermore, that their selected values are different. If $x \equiv y$ then, by definition of equivalence, the true values of the two tests in the current state are equal. Thus, the selected value of one of the

Input: access to $\mathcal{E}$, a finite-state automaton

               D - the diversity of $\mathcal{E}$

               δ - desired confidence

Output: a perfect model of $\mathcal{E}$

Procedure:

```
1  h ← λ
2  δ₀ ← δ/((D − 1)((k + 1)D² + D − 1))
3  initialize T, r, Υ, C and ℓ as in Figure 8 (lines 1-3)
4  repeat
5      execute h, producing output σ
6      if C(t) is incoherent for some t ∈ T then
7          execute t and update C(t)
8      else if ∪ⱼ₌₀ᵐⁱ K(i,j) is incoherent for some 1 ≤ i ≤ ℓ then
9          choose the smallest i for which this is so
10             if K(i,j) is incoherent for some 0 ≤ j ≤ mᵢ then
11                 execute uᵢⱼ and update K(i,j)
12             else [mᵢ = 1, K(i,0) ∩ K(i,1) = ∅ and σ[K(i,0)] ≠ σ[K(i,1)]]
13                 choose j ∈ {0,1} randomly
14                 execute uᵢⱼ and update K(i,j)
15                 if K(i,j) ≠ ∅ then
16                     sᵢ ← sᵢ + 1
17                     if sᵢ > lg(1/δ₀) then
18                         conclude uᵢ₀ ≢ uᵢ₁: update r, T, C, Υ as in Figure 8 (lines 25-31)
19                         ℓ ← 0
20         else
21             make conjecture: handle returned counterexample as in Figure 8 (lines 12-23)
22             sℓ ← −1
23         if K(i,j) = ∅ for some 1 ≤ i ≤ ℓ, 0 ≤ j ≤ mᵢ then
24             h ← huᵢⱼ
25             C(t) ← {0,...,|h|} for t ∈ T
26             ℓ ← 0
27         else if K(i,j) ∩ K(i,j + 1) = ∅ and sᵢ < 0 for some 1 ≤ i ≤ ℓ, 0 ≤ j < mᵢ then
28             sᵢ ← 0
29             uᵢ₀ ← uᵢⱼ; uᵢ₁ ← uᵢ,ⱼ₊₁; mᵢ ← 1
30 end
```

Figure 9: A diversity-based algorithm for inferring $\mathcal{E}$.

candidate sets must disagree with the common value of the two tests in the current state. If this is the case for x (say), and x is executed, then $C(x)$ will be reduced to the empty set (and hx can replace h). In general, if the algorithm *randomly* chooses which of x or y to execute, then with probability $1/2$, the candidate set of the chosen test is emptied. Of course, by repeating such an experiment many times, we can lift our confidence to arbitrarily high levels.

This then is the approach used by our extended algorithm (Figure 9) in determining test inequivalence. The algorithm proceeds just as before. Now, however, when the candidate sets of two tests are found to be disjoint (line 23), the procedure does not immediately conclude that the tests are inequivalent. Rather, it keeps the two candidate sets around and, when given the opportunity, re-runs the two tests as described above. Only after the tests have been re-run many times with neither of the candidate sets emptying does the algorithm conclude that the tests are inequivalent.

Theorem 5.2 *The algorithm of Figure 9, with probability at least $1 - \delta$, halts and outputs a perfect model. The algorithm's running time, and the number of actions executed are both bounded by*

$$O(kD^4(m + D)(mD + \log(kD/\delta))).$$

Proof: The proof of this theorem is quite similar to the proof of Theorem 5.1. As before, we need only show that the algorithm halts in the stated number of steps since it only halts when a perfect model has been found.

As before, $i \notin C(t)$ only if $h_i \not\equiv ht$ for any test t and $0 \leq i \leq |h|$. Similarly for $K(i,j)$.

In the algorithm, the variable s_i serves two purposes: When the sequence of tests $u_{i0}, \ldots, u_{im_i}$ is first created (lines 21-22), s_i is set to -1. Variable s_i remains negative until the candidate sets of two consecutive tests in this sequence are reduced to the point that they are disjoint. At this point, s_i becomes a (non-negative) counter indicating how many times the tests u_{i0} and u_{i1} have been rerun without either candidate set emptying. It is easily verified that, on each iteration, $K(i,j) \cap K(i,j+1) \neq \emptyset$ for $0 \leq j < m_i$ if $s_i < 0$, and $m_i = 1$ and $K(i,0) \cap K(i,1) = \emptyset$ if $s_i \geq 0$. Note that this implies at line 8 that $\bigcup_j K(i,j)$ is coherent if and only if every $K(i,j)$ is coherent and $K(i,0)$ and $K(i,1)$'s selected values agree.

The algorithm is randomized, and can only be shown to behave correctly when certain low probability events do not occur. Therefore, to simplify the analysis, we will assume that the algorithm has a *good run* — specifically, that if $u_{i0} \equiv u_{i1}$ then s_i does not exceed $\lg(1/\delta_0)$, for $1 \leq i \leq \ell$. Later, we will show that a good run occurs with probability at least $1 - \delta$.

Assuming then that a good run occurs, it is clear that $t \notin r(x)$ only if $x \not\equiv t$ for $t \in T$, $x \in BT$. Thus, all pairs of tests in T are inequivalent, and $|T| \leq D$. Further, this shows that lines 18-19 are executed no more than $(k + 1)D^2$ times.

As argued above, h cannot be extended more than $D - 1$ times, implying that lines 24-26 are executed at most $D - 1$ times. Thus, variable ℓ is reset to zero no more than $(k + 1)D^2 + D - 1$ times. Later we will again argue that $\ell \leq D - 1$ on each iteration; assume for now that this is the case. Then since $s_i \leq \lg(1/\delta_0)$ on each iteration, lines 13-19 are executed at most $(D - 1)((k + 1)D^2 + D - 1)\lg(1/\delta_0)$ times.

It can be verified, as in Theorem 5.1, that the set $\{0 \leq i \leq |h| : h_i \equiv x\}$ respects $C(t)$ for any tests t and x, on each iteration (and likewise for $K(i,j)$). Applying our bound on ℓ and on the number of times ℓ is reset, this implies line 11 is executed at most $(D-1)^2 m((k+1)D^2 + D - 1)$ times, and so the condition at line 8 is satisfied at most $(D-1)((k+1)D^2 + D - 1)((D-1)m + \lg(1/\delta_0))$ times.

The sets $C(t)$ for $t \in T$ are reset to $\{0, \ldots, |h|\}$ at most $D - 1$ times (i.e., only when h is extended). Thus, the condition at line 6 is satisfied at most $D(D-1)^2$ times.

Finally, since ℓ is bounded and is executed at line 21, the conditions at lines 6 and 8 fail to be satisfied at most $(D-1)((k+1)D^2 + D - 1)$ times. Thus, the number of iterations of the outer loop can be computed, and the bound on the number of actions executed follows from the fact that $|h| \leq (D-1)(m + D - 1)$.

The proof that $\ell \leq D - 1$ is quite similar to that given in the proof of Theorem 5.1. As before, we define graphs $G_0, \ldots, G_\ell$ on vertex set $\{0, \ldots, |h|\}$. We let $\{r, s\}$ be an edge of G_i if and only if $h_r \equiv h_s$ or $\{r, s\} \subset \bigcup_{j=0}^{m_{i'}} K(i', j)$ for some $1 \leq i' \leq i$. Then G_0 has at most D connected components. It can be argued as before that $\bigcup_j K(i', j)$ is respects $K(i, j')$ if $i' < i$. Also, $K(i, 0)$ and $K(i, m_i)$ are disjoint. Therefore, if r is in $K(i, 0)$ and s is in $K(i, m_i)$ then r and s are connected in G_i but not in G_{i-1} by the argument given in the proof of Theorem 5.1. Thus, G_{i-1} has at least one more connected component than G_i, and $\ell \leq D - 1$.

Thus, we have proved that the stated bound on the number of actions executed holds on a good run. It remains then only to show that a good run occurs with probability at least $1 - \delta$.

As argued above, if $K(i, 0)$ and $K(i, 1)$ are coherent with different selected values, and if $u_{i0} \equiv u_{i1}$, then the probability is $1/2$ that $K(i, j)$ is empty after u_{ij} is executed, for j chosen randomly from $\{0, 1\}$. Thus, if $u_{i0} \equiv u_{i1}$, then the probability that s_i exceeds k is less than 2^{-k}. particular. s_i exceeds $\lg(1/\delta_0)$ with probability less than δ_0.

We argued above that. on a good run, lines 21–22 are executed at most $(D-1)((k+1)D^2 + D - 1)$ times: that is. at most this many pairs u_{i0}, u_{i1} are created. The chance that s_i exceeds $\lg(1/\delta_0)$ when $u_{i0} \equiv u_{i1}$ for any of these pairs is thus bounded by δ. Thus, δ bounds the probability of a bad run. completing the proof of the action execution bound.

It is clear that this algorithm runs in polynomial time. It is not so obvious, however, how it can be implemented to run in time proportional to the bound on the number of actions executed. We discuss techniques that can be used to achieve such a time bound.

Perhaps the most time consuming task performed by the algorithm is in checking the coherence of the many candidate sets. In a naive implementation, determining the coherence of a subset of $\{0, \ldots, |h|\}$ takes $O(|h|)$ time. Thus, for instance, checking the $|T|$ candidate sets $C(t)$ at line 6 takes up to $O(D|h|)$ time; since only $O(|h| + D + m)$ actions are executed on each iteration, this gives a time bound that exceeds the action execution bound by at least a factor

of D.

We show instead how the coherence of any candidate set can be checked in $O(D)$ time using a different representation: We maintain a partition π of the set $\{0,\ldots,|h|\}$ with the interpretation that i and j are in the same block of π if and only if every time that h was previously executed (line 5), it was observed that $\sigma_i = \sigma_j$ (where σ was the observed output sequence, as usual). In particular, if $h_i \equiv h_j$, then i and j are always in the same block of π. Thus $|\pi| \leq D$.

Note that if such a partition is maintained, then on each iteration, each block of π respects each candidate set $C(t)$ or $K(i,j)$. It therefore makes sense to represent each candidate set as a set of pointers to the blocks of π that it includes. If this is done, then each candidate set contains at most D pointers, and each set's coherence can be determined in $O(D)$ time (it is only necessary to examine the value of one member of each block since all the other members have the same value).

It is quite easy to see how the partition π can be maintained: Initially, and each time h is extended, π is set to $\{\{0,\ldots,|h|\}\}$. After h is executed with output σ at line 5, the coherence of each block of π is determined. Since each index $0,\ldots,|h|$ occurs in only one block of π, this only takes $O(|h|)$ time. If any block s is incoherent, then it is split into two new blocks $s \cap \sigma^{-1}(0)$ and $s \cap \sigma^{-1}(1)$. Since $|\pi| \leq D$, this can happen at most $D - 1$ times. Naturally, when it does happen, all of the candidate sets must be changed so that their members point to blocks of the new partition. This takes $O(D)$ time for each of the $O(mD)$ candidate sets. Thus, since h can be extended at most $D - 1$ times, the algorithm spends at most $O(mD^4)$ time updating candidate sets in this fashion. (This time is negligible compared to the number of actions executed.)

This still does not give the desired time bound because, even with this modification, naively computing ℓ unions, each of up to m candidate sets as at line 8, can take $O(mD^2)$ time. Instead of the naive approach, we therefore maintain a counter $e(i,s)$ for each $1 \leq i \leq \ell$ and $s \in \pi$. This counter indicates the number of candidate sets $K(i,j)$ which include s: $e(i,s) = |\{0 \leq j \leq m_i : s \subset K(i,j)\}|$. It is straightforward how such a counter can be efficiently maintained, and the union $\bigcup_j K(i,j)$ can now be easily computed in $O(D)$ time as the union of those blocks $s \in \pi$ for which $e(i,s) > 0$.

Finally, lines 23 and 27, which appear to require a great deal of search, actually do not because only a small number of values i, j (those for which $K(i,j)$ was modified) need actually be checked.

With these ideas, it can now be fairly easily verified that the algorithm halts within the stated time bound. ∎

5-6   A state-based algorithm for permutation automata

In this and the next sections, we present algorithms for inferring permutation automata. Unlike the procedures described up to this point, these procedures do *not* rely on a means of discovering counterexamples; the procedures actively experiment with the unknown environment, and output a perfect model with arbitrarily high probability.

As before, we describe both a state-based and a diversity-based procedure. In both cases, we describe deterministic procedures that, given a (diversity-based) homing sequence h, will output a perfect model of the environment in time polynomial in n (or D) and $|h|$. To construct the needed homing sequence, we show that any sufficiently long random sequence of actions is likely to be a homing sequence.

We begin in this section with the state-based case. Consider first the simpler problem of inferring a *visible* automaton, i.e., one in which the identity of each state is readily observable. For instance, suppose each state, instead of outputting 0 or 1, outputs its own name. In this situation, inference of the automaton is almost trivial. From the current state q, we can immediately learn the value of $\delta(q, b)$ by simply executing b and observing the state reached. If $\delta(q, b)$ is already known for all the basic actions, then either we can find a path based on what is already known about δ to a state for which this is not the case, or we have finished exploring the automaton. It is not hard to see that $O(kn^2)$ actions are executed in total by this procedure.

Now suppose that the unknown environment $\mathcal{E}$ is a permutation automaton and that a homing sequence h has been provided. Because $\mathcal{E}$ is a permutation environment, we can easily show that h is also a *distinguishing sequence*, that is, h distinguishes every pair of unequal states of $\mathcal{E}$. Put another way, $q_1\langle h \rangle = q_2 \langle h \rangle$ if and only if $q_1 = q_2$. (For if $q_1 \langle h \rangle = q_2 \langle h \rangle$ then, since h is a homing sequence, $q_1 h = q_2 h$. This implies $q_1 = q_2$ since $\mathcal{E}$ is a permutation environment.) Thus, the identity of any state is uniquely given by the output of h at that state; its identity is almost directly observable.

To infer the environment, we therefore use the inference procedure sketched above for visible automata. Each state q is named or represented by $q\langle h \rangle$, the output of h at that state. To identify the current state, simply execute h and observe the output produced.

Although executing h is helpful in identifying the state from which the sequence was executed, doing so is also likely to leave us in a state at the end of the sequence whose identity is unknown. This is a problem because the visible-automaton inference procedure requires that we be able to find a state whose identity is known even without executing h. We can overcome this problem, however, by maintaining a table u which records the fact that if $\sigma = q\langle h \rangle$ was just observed as the output of executing h, then the output of h if executed from the current state qh is given by $u(\sigma)$.

Input: access to $\mathcal{E}$, a permutation automaton
              h - homing sequence
Output: a perfect model of $\mathcal{E}$
Procedure:
1 d, u are initially undefined everywhere
2 execute h, producing output σ
3 **repeat**
4   **if** $u(\sigma)$ is not defined **then**
5   execute h, producing output τ
6          $u(\sigma) \leftarrow \tau$
7          $\sigma \leftarrow \tau$
8   **else if** $(\exists a \in A, b \in B)$ $d(u(\sigma), a)$ is defined, but $d(u(\sigma), ab)$ is undefined **then**
9   choose the shortest such ab
10          $\alpha \leftarrow d(u(\sigma), a)$
11   execute ab
12   execute h, producing output τ
13          $d(\alpha, b) \leftarrow \tau$
14          $\sigma \leftarrow \tau$
15   **else**
16   exit loop
17 **end**
18 let q be the current state
19 **output** the following prediction rule (model of $\mathcal{E}$):
20   on input $a \in A$,
21          $\alpha \leftarrow d(u(\sigma), a)$
22   predict $\gamma(qa) = \alpha_0$

Figure 10: A state-based algorithm for inferring permutation environment $\mathcal{E}$.

Thus, we can reach a state whose identity is known (without executing h from it), we can execute an experiment as dictated by the visible-automaton inference procedure, and we can identify the last state reached by executing h. This can of course be repeated as many times as necessary.

Our procedure is given in Figure 10. As mentioned, each state q is represented by $q\langle h \rangle$, the output of h at q. For $\sigma \in Q\langle h \rangle$, we write q_σ to denote that state for which $q_\sigma\langle h \rangle = \sigma$. This state is well-defined since h is a distinguishing sequence. A function or table $u : Q\langle h \rangle \to Q\langle h \rangle$ is maintained for which $u(\sigma) = q_\sigma h\langle h \rangle$. That is, if h was just executed with output σ, then the current state is $q_{u(\sigma)}$.

The transition function is represented by the program variable $d : Q\langle h \rangle \times B \to Q\langle h \rangle$. For notational purposes, the function d can be extended in the usual manner to the domain $Q\langle h \rangle \times A$. The variable d is used to store and compute the output of h in future states. Given $\sigma \in Q\langle h \rangle$ and $b \in B$, $d(\sigma, b)$ denotes the output of h in state $q_\sigma b$. That is, if properly constructed,

$d(\sigma, b) = q_\sigma b\langle h\rangle$.

Theorem 6.1 *The algorithm of Figure 10 halts and outputs a perfect model of $\mathcal{E}$ after executing at most $O(kn(|h| + n))$ actions, and in time $O(kn(|h| + kn))$.*

Proof: Clearly, $|Q\langle h\rangle| \le n$, so that after at most $n + kn$ iterations, the procedure will halt since every entry of u and d will be defined.

We can view d as defining a directed graph whose vertices are the elements of $Q\langle h\rangle$, and whose edges are of the form $\sigma \to d(\sigma, b)$ whenever $\sigma \in Q\langle h\rangle$, $b \in B$ and $d(\sigma, b)$ is defined. Then the problem of finding an experiment ab as in the figure can be treated as that of finding a path in the graph from $u(\sigma)$ to another vertex α whose out-degree is less than k. This is easily done in $O(kn)$ time (for instance, using breadth-first search), and the resulting experiment ab has length at most n, the size of the graph. This proves the upper bound on the number of actions executed.

The remaining steps of the loop can be achieved in $O(|h|)$ time, for instance, if we store the elements of $Q\langle h\rangle$ at the leaves of a depth $(|h| + 1)$ binary tree. It remains then only to show that the prediction rule output by the algorithm is a perfect model of $\mathcal{E}$.

We prove this by showing that the following invariants hold between each iteration of the main loop:

1. If $\sigma \in Q\langle h\rangle$ and $u(\sigma)$ is defined, then $u(\sigma) = q_\sigma h\langle h\rangle$.

2. If $\sigma \in Q\langle h\rangle$, $b \in B$ and $d(\sigma, b)$ is defined then $d(\sigma, b) = q_\sigma b\langle h\rangle$.

Initially, these invariants hold vacuously since u and d are undefined everywhere. Suppose at the top of an iteration of the loop that h was just executed from some state q with output σ. Then $q = q_\sigma$, and the current state is $q_\sigma h$. If $u(\sigma)$ is undefined, then h is executed from the current state with output τ. Thus, we learn that $\tau = q_\sigma h\langle h\rangle$. Setting $u(\sigma)$ to τ, invariant 1 is maintained.

On the other hand, if $u(\sigma)$ is defined, then the current state is $q_{u(\sigma)}$. If an experiment ab is found as shown in the figure, then invariant 2, together with an easy induction argument on the length of a, shows that $\alpha = d(u(\sigma), a) = q_{u(\sigma)} a\langle h\rangle$. The state we reach by executing a then is just q_α. Executing b, and then h with output τ, we learn that $\tau = q_\alpha b\langle h\rangle$. Setting $d(\alpha, b) = \tau$, invariant 2 is maintained.

With these invariants, it is not hard to see why, after the loop is exited, the output prediction rule is correct. The current state q is just $q_{u(\sigma)}$ as before. Given $a \in A$, we have $qa\langle h\rangle = d(u(\sigma), a)$. Therefore, the first element of $d(u(\sigma), a)$ is $\gamma(qa)$. ∎

Finally, we must consider how to construct h. In fact, any sufficiently long random sequence of actions is likely to be a homing sequence:

Theorem 6.2 *Let $\delta > 0$, and let h be a random sequence of length $8kn^5 \cdot \ln(n) \cdot (n + \ln(1/\delta))$. Then h is a homing sequence with probability at least $1 - \delta$.*

Proof: The idea is to randomly construct the homing sequence in the manner described in Figure 3. On each iteration, an appropriate extension x which distinguishes some pair of states as needed by the algorithm is likely to be given by any sufficiently long random walk. This follows from previous results on random walks in permutation automata. Specifically, we will use the following result:

Lemma 6.3 *Let q_1 and q_2 be two distinct states and let x be a random sequence of length $2kn^4 \ln(n)$ of the following form: At each step, with equal probability, we either do nothing, or we execute a uniformly and randomly chosen basic action from B. Then the probability that $\gamma(q_1 x) \neq \gamma(q_2 x)$ is at least $1/2n$.*

Essentially, the same result is proved in my master's thesis [79] using results of Fiedler [24] on the eigenvalues of doubly stochastic matrices, in addition to certain properties of point-symmetric graphs. There, the result was proved for update graphs, but, because of the "dual" relationship between update graphs and finite automata, the results holds as stated as well.

Let $x_1, \ldots, x_r$ be a sequence of random strings, each of length $2kn^4 \ln(n)$. Let $y_i = x_1 x_2 \cdots x_i$. We wish to show that y_r is a homing sequence with high probability. Consider a sequence of trials in which "success" on the ith trial means that either y_{i-1} is a homing sequence (so that y_i is as well) or $|Q\langle y_i \rangle| > |Q\langle y_{i-1} \rangle|$. Clearly, if n of the trials succeed, then y_r is a homing sequence.

For any choice of y_{i-1}, Lemma 6.3 shows that the probability of success on the ith trial is at least $1/2n$. Thus, applying Chernoff bounds (Lemma 2-3.6), we see that the probability of fewer than n successes in r trials is at most δ if $r \geq 4n(n + \ln(1/\delta))$. This proves the theorem. ∎

These theorems give our inference procedure a running time of $O(k^2 n^6 \log(n) \cdot (n + \log(1/\delta)))$.

5-7   A diversity-based algorithm for permutation automata

We can show in a similar manner how a permutation environment can be inferred using a diversity-based representation. As before, we reduce the problem to that of inferring a visible automaton — in this case, one for which all of the test-equivalence classes are known, and for which the value of each test class is observable in every state. The problem of inferring such automata is solved in Chapter 4 of my master's thesis [79]; the solution is based on the careful planning of experiments, and on the maintenance of candidate sets similar to those described in Section 5-5.

Let h be a given diversity-based homing sequence for the unknown permutation environment $\mathcal{E}$. As before, to simulate the inference algorithm for visible automata, it suffices to show that the state of the automaton (i.e. the values of the test classes) can be observed by executing h, and further that it is possible to reach a state whose identity is known even without executing h. Since $\mathcal{E}$ is a permutation environment, we can show that every test class is represented by some prefix of h. Therefore, at the current state q, the values of all the test classes can be observed simply by executing h.

If, having executed h from some state q, we find that candidate set $C(h_i)$ is coherent, then the value of test h_i in the current state qh is just the selected value of $C(h_i)$. (As before, h_i is the prefix of h of length i.) Thus, if all the candidate sets are coherent, then $qh\langle h \rangle$, the output of the entire sequence, is known in the current state. On the other hand, if one of the candidate sets is incoherent, then by re-executing h we are guaranteed to reduce one of the candidate sets. Thus, we can quickly reach a state in which the output of h is known without actually executing it.

We say action sequence a is a *diversity-based distinguishing sequence* if every test is equivalent to some prefix of a. Such a sequence is clearly a distinguishing sequence, since if $q_1 \neq q_2$ then there exists a test t distinguishing the two states; since $t \equiv p$ for some prefix p of a, $\gamma(q_1 p) \neq \gamma(q_2 p)$ and so $q_1\langle a \rangle \neq q_2\langle a \rangle$.

A diversity-based distinguishing sequence is also a diversity-based homing sequence, as is obvious from their definitions. In permutation environments (but not in general), the converse holds: Suppose h is a diversity-based homing sequence. Let $[t_1], [t_2], \ldots [t_D]$ be the equivalence classes of $\mathcal{E}$. Then there exist prefixes $p_1, p_2, \ldots p_D$ of h such that $p_i \equiv ht_i$. Since $\mathcal{E}$ is a permutation environment, if $t_i \not\equiv t_j$ then $ht_i \not\equiv ht_j$. Therefore, the D prefixes p_i are pairwise inequivalent, and so every equivalence class is represented by some prefix of h. Thus, h is a diversity-based distinguishing sequence.

As in the last section, we assume a diversity-based homing sequence h has been given, and show later how such a sequence can be randomly constructed.

Our procedure is given in Figure 11. As mentioned above, the algorithm maintains various kinds of candidate sets. First, for each $0 \leq i \leq |h|$, a set $G(i)$ is maintained with the interpretation that j is in $G(i)$ if h_j could plausibly be equivalent to hh_i, i.e., if it has not yet been determined that $h_j \not\equiv hh_i$. (Thus, $G(i) = C(h_i)$ in the notation of Section 5-5.) As described above, such candidate sets are useful for reaching a state in which the output of h is known prior to its execution from that state.

The algorithm also maintains sets $U(i, b)$ for $0 \leq i \leq |h|$ and $b \in B$; these sets consist of indices j for which h_j is plausibly equivalent to bh_i. To see why such sets might be useful, suppose h has been executed from some state q with output σ. As seen before, if $G(i)$ is

Input: access to $\mathcal{E}$, a permutation automaton
           h - a diversity-based homing sequence
Output: a perfect model of $\mathcal{E}$
Procedure:
1 $G(i), U(i,b) \leftarrow \{0, \ldots, |h|\}$ **for** $i \in \{0, \ldots, |h|\}, b \in B$
2 execute h, producing output σ
3 **repeat**
4   **if** $G(i)$ is incoherent for some $0 \le i \le |h|$ **then**
5   execute h, producing output τ
6              $G(i) \leftarrow G(i) \cap \sigma^{-1}(\tau_i)$ **for** $i \in \{0, \ldots, |h|\}$
7              $\sigma \leftarrow \tau$
8   **else**
9              $\beta_i \leftarrow \sigma[G(i)]$ **for** $i \in \{0, \ldots, |h|\}$
10   **if** PLAN-EXP can find a shortest useful experiment ab **then**
11                   $\alpha_i \leftarrow \beta[U(i,a)]$ **for** $i \in \{0, \ldots, |h|\}$
12   execute ab
13   execute h, producing output τ
14                   $U(i,b) \leftarrow U(i,b) \cap \alpha^{-1}(\tau_i)$ **for** $i \in \{0, \ldots, |h|\}$
15                   $\sigma \leftarrow \tau$
16   **else**
17   exit loop
20 **end**
21 let q be the current state
22 $\beta_i \leftarrow \sigma[G(i)]$ **for** $i \in \{0, \ldots, |h|\}$
23 **output** the following prediction rule (model of $\mathcal{E}$):
24   on input $a \in A$, predict $\gamma(qa) = \beta[U(0,a)]$

Figure 11: A diversity-based algorithm for inferring permutation environment $\mathcal{E}$.

coherent for all i, then $\beta = qh\langle h \rangle$ is known, the output of h if executed from the current state qh. If, moreover, $U(i,b)$ is coherent (with respect to β), then $\gamma(qhbh_i)$ is known; thus, if this is the case for all i, then $qhb\langle h \rangle$ can be determined, the output of h from the state reached if b were executed.

The function U can be extended in a natural manner to the domain $\{0, \ldots, |h|\} \times A$ by the rule $U(i, \lambda) = \{i\}$ and $U(i, ab) = \bigcup_{j \in U(i,a)} U(j,b)$ for $i \in \{0, \ldots, |h|\}$, $a \in A$ and $b \in B$. Then the above statements also hold if b is replaced by any action $a \in A$.

Our algorithm works by trying to reduce the candidate sets $U(i,b)$ as much as possible until $U(i,a)$ is coherent for all i and all $a \in A$; at this point, from the preceding comments, a perfect model has been attained.

Let σ, β and q be as above, assuming all $G(i)$'s are coherent with respect to σ. If $U(i,b)$ is incoherent (with respect to β), then executing b and then h will clearly cause some candidate

set $U(i, b)$ to shrink. In this case, b is called an *immediately useful experiment*. However, it may be the case that there is no immediately useful experiment (all the sets $U(i, b)$ are coherent) but, nevertheless, some set $U(i, a)$ is incoherent for $a \in A$ so that a perfect model has not been achieved. In this case, it is possible to find a *useful experiment*; this is an experiment in which a "set-up" action $a \in A$ is first executed leading to a state in which an immediately useful experiment can be executed.

More precisely, a sequence ab, where $a \in A$ and $b \in B$, is a *useful experiment* if, for some $0 \le i \le |h|$, $U(i, ab)$ is incoherent, but $U(j, a)$ is coherent for $j \in U(i, b)$. Note that the *shortest* useful experiment has the additional property that $U(j, a)$ is coherent for *all* j, $0 \le j \le |h|$ (otherwise, a prefix of a would be a shorter useful experiment). A procedure for finding a shortest useful experiment, called PLAN-EXP, was described in my master's thesis [79], and is treated here as a "black-box" subroutine. (The inputs required by PLAN-EXP are omitted from Figure 11, but are described fully below.)

Thus, at a high level, our algorithm is simple: execute h; if some $G(i)$ is incoherent, then re-execute h and update G; otherwise, find and execute a shortest useful experiment, and update U. If no useful experiment exists, then a perfect model has been found.

Theorem 7.1 *The algorithm described in Figure 11 halts and outputs a perfect model of $\mathcal{E}$ after executing at most $O(kD(|h| + D))$ actions, and in time $O(kD(|h| + D^2 + kD \cdot \alpha(kD, D)))$.*

Proof: First, note that because of the manner in which G is updated, an index j is removed from $G(i)$ only if $h_j \not\equiv hh_i$. Thus, since h is a diversity-based homing sequence, $G(i)$ is never empty. Also, if j is removed from $G(i)$, then every other index j' for which $h_j \equiv h_{j'}$ must also be removed since equivalent tests have the same value in every state.

In addition, every index j appears in some set $G(i)$, i.e., $\bigcup_i G(i) = \{0, \ldots, |h|\}$. To see that this is so, note that, because h is a diversity-based distinguishing sequence, every equivalence class is represented by the prefixes of h, that is, $|\{[h_i] : 0 \le i \le |h|\}| = D$. Since $\mathcal{E}$ is a permutation environment, $h_i \equiv h_j$ if and only if $hh_i \equiv hh_j$. Thus, $|\{[hh_i] : 0 \le i \le |h|\}| = D$. Therefore, the test h_j is equivalent to some hh_i, implying $j \in G(i)$.

For the analysis, it is important to note that the set $\{G(i) : 0 \le i \le |h|\}$ is a partition of $\{0, \ldots, |h|\}$. This can be proved by an inductive argument: For suppose, prior to the execution of line 6, that $G(i)$ and $G(j)$ are equal or disjoint, for some i, j. Then if $G(i) = G(j)$ and $\tau_i = \tau_j$, then clearly $G(i) \cap \sigma^{-1}(\tau_i) = G(j) \cap \sigma^{-1}(\tau_j)$. On the other hand, if $G(i)$ and $G(j)$ are disjoint or if $\tau_i \ne \tau_j$, then $G(i) \cap \sigma^{-1}(\tau_i)$ and $G(j) \cap \sigma^{-1}(\tau_j)$ must also be disjoint. In either case, the new sets following the execution of line 6 will be equal or disjoint.

Thus, since the set $\{0 \le i \le |h| : h_i \equiv x\}$ respects $G(i)$ for any test x on each iteration, it follows that $|\{G(i) : 0 \le i \le |h|\}| \le D$ on each iteration. Since, some $G(i)$ shrinks each time that lines 5–7 are executed, it follows that this block is executed at most $D - 1$ times.

We would like to give a similar argument showing that lines 9–17 are executed at most $k(D-1)$ times. We will first give an inductive proof that $j \notin U(i,b)$ only if $h_j \not\equiv bh_i$:

Suppose that h has been executed from q with output σ. Suppose also that each $G(i)$ is coherent with respect to σ. Then there is some j for which $h_j \equiv hh_i$, and which is therefore in $G(i)$. Thus $\gamma(qhh_i) = \gamma(qh_j) = \sigma_j = \sigma[G(i)]$, and so $qh\langle h \rangle = \beta$ where $\beta_i = \sigma[G(i)]$, as in the figure.

Suppose that PLAN-EXP returns an experiment ab. Then $U(i,a)$ is coherent (with respect to β) for all $0 \le i \le |h|$, but, for some i, $U(i,ab)$ is not. Since h is a diversity-based distinguishing sequence, there exists j for which $h_j \equiv ah_i$. By inductive hypothesis, $j \in U(i,a)$. Since $U(i,a)$ is coherent, we have $\alpha_i = \beta[U(i,a)] = \gamma(qhh_j) = \gamma(qhah_i)$. Thus, $\alpha = qha\langle h \rangle$.

It can now be verified that j is removed from $U(i,b)$ only if $h_j \not\equiv bh_i$, completing the induction. As before, this implies that each $U(i,b)$ is nonempty on each iteration, and that $\bigcup_i U(i,b) = \{0,\ldots,|h|\}$ on each iteration for each $b \in B$. Also, having argued that $\alpha = qha\langle h \rangle$ at this point in the program, it can now be argued as before that the set $\{i : h_i \equiv x\}$ respects each $U(i,b)$, and that the set $\{U(i,b) : 0 \le i \le |h|\}$ is a partition consisting of at most D blocks for each $b \in B$.

Since ab is a useful experiment, some set $U(i,b)$ must shrink at line 14. Thus, by the preceding arguments, lines 9–17 are executed at most $k(D-1)$ times.

We will later argue that the returned useful experiment has length at most D. This will then complete the proof of the action execution bound.

Given the above arguments, it is quite easy to prove the correctness of the output prediction rule: On exiting the main loop, each set $G(i)$ or $U(i,a)$ is coherent (with respect to σ and β, respectively, as in the figure) for all i and $a \in A$. As argued above, in the current state q, this implies that $q\langle h \rangle = \beta$. Also, given $a \in A$, we showed above that $\gamma(qah_i) = \beta[U(i,a)]$. Thus, $\gamma(qa) = \beta[U(0,a)]$, and the output rule is a perfect model.

Finally, we turn to efficiency considerations. If naively implemented, the running time of the procedure may be quite poor. However, using similar techniques to those described in Section 5-5, we can derive a time bound comparable to the action execution bound.

In particular, we maintain a partition π over the set $\{0,\ldots,|h|\}$ with the condition that i and j belong to the same block of π if and only if the values of h_i and h_j have never differed on any execution of h (so that the two tests are plausibly equivalent). As before, if $h_i \equiv h_j$, then i and j must be in the same block of π. Thus, $|\pi| \le D$.

It is easily verified that, on each iteration, if i and i' are in the same block of π, then $G(i) = G(i')$, and $\{i,i'\}$ respects each set $G(j)$. Similarly, for $b \in B$, $U(i,b) = U(i',b)$ and $\{i,i'\}$ respects each set $U(j,b)$. Thus, with respect to the data structures G and U, the two indices i and i' are entirely indistinguishable. Therefore, we can represent these structures more

efficiently in terms of the blocks of π.

In particular, as was done in Section 5-5, we can represent each candidate set as a list of pointers to those blocks of π which it includes. Thus, the representation of such a set has size at most D. Also, since $G(i) = G(j)$ if i and j are in the same block, we only need maintain a candidate set for a single member of each block (say, the minimum element). That is, we maintain a candidate set $G(i)$ or $U(i,b)$ (explicitly represented as described above) if and only if i is the smallest member of its block; the other candidate sets are only implicitly maintained, based on the equalities among candidate sets described above.

With such a representation, lines 4, 6 and 14 take only time $O(D^2)$. Using the fact (to be proved) that $|ab| \leq D$, we can also show that line 11 takes time $O(D^2 + |h|)$: computing $\alpha_i = \beta[U(i,a)]$ for a single value of i takes $O(D)$ time since $|a|$ is bounded, and since $U(i,a)$ is known to be coherent. Thus, computing α_i for each $i \in \{\min(s) : s \in \pi\}$ takes $O(D^2)$ time. Finally, all the other values of α_i can be computed by setting $\alpha_i \leftarrow \alpha_{\min(s)}$ for $s \in \pi, i \in s$ in $O(|h|)$ time.

The partition π is easily maintained in the same manner described in Section 5-5: Each time that h is executed, the coherence of each block of π is checked in $O(|h|)$ time. If any block is incoherent, then the structures G and U must be updated; this takes $O(kD^2)$ time. Since π can be partitioned at most D times, this adds $O(kD^3)$ to the total running time of the procedure.

It remains then only to show how the running time of PLAN-EXP can be bounded. The procedure PLAN-EXP takes as input a set V of variables; a set of candidate sets for each $v \in V$, $v \in B$; and an assignment to the variables in V. It returns a shortest useful experiment ab (or reports that none exists) in time $O(k|V| \cdot \alpha(k|V|, |V|))$ where α is a functional inverse of Ackermann's function [82]. The length of the returned experiment ab is bounded by $|V|$.

Thus, if we use $\{0, \ldots, |h|\}$ as our variable set in our call to PLAN-EXP, then the procedure may take too long, and could plausibly return an experiment far longer than D. Instead, we will use the blocks of π as our variable set. The candidate sets are then defined naturally by the rule $U''(s,b) = \{s' \in \pi : s' \subset U(\min(s),b)\}$ for $s \in \pi$ and $b \in B$. The assignment $\beta'(s)$ is similarly defined to be $\beta(\min(s))$.

Note that our representation scheme for U is essentially equivalent to the structure U', and the structure β' is easily computed in $O(D)$ time. Also, since $|\pi| \leq D$, PLAN-EXP runs in time $O(kD \cdot \alpha(kD, D))$, and returns an experiment of length at most D.

It can be argued by induction on the length of a that $U''(s,a) = \{s' \in \pi : s' \subset U(i,a)\}$ for $s \in \pi$ and $a \in A$, assuming $i \in s$. With this fact, it can be seen that $U''(s,a)$ is coherent with respect to β' if and only if $U(i,a)$ is coherent with respect to β.

In particular, this shows that if PLAN-EXP when called in this manner returns an experiment ab, then $U(i,a)$ is coherent (with respect to β) for all $0 \leq i \leq |h|$, but, for some i, $U(i,ab)$

is not; that is, ab is indeed a shortest useful experiment. Likewise, if PLAN-EXP fails to find a useful experiment, then each $U(i,a)$ is coherent for all i and all $a \in A$.

This completes the proof.   ∎

As in the state-based case, we can construct a diversity-based homing sequence by choosing a sufficiently long sequence of actions. Below, $H_n = \sum_{i=1}^{n}(1/i)$ is the nth harmonic number. It is well known that $H_n = \theta(\log n)$.

Theorem 7.2 *Let $\delta > 0$, and let h be a random sequence of length $2kD^3 H_D \cdot \ln(D) \cdot \ln(D/\delta)$. Then h is a diversity-based homing sequence with probability at least $1 - \delta$.*

Proof: We follow the algorithm of Figure 4 for constructing a diversity-based homing sequence. On each iteration, we need to find an extension x to h for which hx is inequivalent to every prefix of h. That is, if v equivalence classes are represented by the prefixes of h, and $[t_1], [t_2], \ldots, [t_{D-v}]$ are the equivalence classes *not* represented, then we wish to find x such that $hx \equiv t_i$ for some i. Equivalently, we want $x \equiv h^{-1}t_i$. (Here, h^{-1} is a sequence of actions for which $h^{-1}h$ is the "identity" action, i.e., $qh^{-1}h = q$ for all $q \in Q$. The existence of h^{-1} is guaranteed by the fact that $\mathcal{E}$ is a permutation environment.)

Based on the results on random walks given in my master's thesis [79], it is easy to conclude the following:

Lemma 7.3 *Let t be any test, and let x be a random sequence of length $kD^2 \ln(D)$ of the following form: At each step, with equal probability, we either do nothing, or we execute a uniformly and randomly chosen basic action from B. Then the probability that $t \equiv x$ is at least $1/2D$.*

Thus, the probability that an extension x as described above will be equivalent to any $h^{-1}t_i$ is at least $(D - v)/2D$. Extending h in this manner $(2D/(D - v)) \cdot \ln(1/\delta)$ times gives a probability of at least $1 - \delta$ of successfully increasing the number of equivalence classes represented by the prefixes of h. Replacing δ with δ/D, we can conclude that h is a homing sequence with probability at least $1 - \delta$ if its length is at least

$$\sum_{v=0}^{D-1} kD^2 \ln(D) \cdot \frac{2D}{D - v} \cdot \ln(D/\delta)$$

as claimed. (This sequence may be longer than strictly necessary since v may increase by more than one with each extension; also, many of the "actions" required by the lemma are actually "no-ops." This, however, does not affect the argument since a homing sequence remains one even if suffixed or prefixed.)   ∎

Thus, our inference procedure runs in time $O(k^2 D^4 \log^2(D) \cdot \log(D/\delta))$. This improves the previously best-known bound of $O(k^2 D^7 \log(D) \cdot \log(kD/\delta))$ given by Rivest and Schapire [73, 79] by roughly a factor of $D^3/\log(D)$.

5-8 Experimental results

The algorithm described in Section 5-4 has been implemented and tested on several simple robot environments.

In the "Random Graph" environment, the robot is placed on a randomly generated directed graph. The graph has n vertices, and each vertex has one out-going edge labeled with each of the k basic actions. For each vertex i, one edge (chosen at random) is directed to vertex $i + 1 \bmod n$; this ensures that the graph contains a Hamiltonian cycle, and so is strongly connected. The other edges point to randomly chosen vertices, and the output of each vertex is also chosen at random.

In the "Knight Moves" environment, the robot is placed on a square checker-board, and can make any of the legal moves of a chess knight. However, if the robot attempts to move off the board, its action fails and no movement occurs. The robot can only sense the color of the square it occupies. Thus, when away from the walls, every action simply inverts the robot's current sensation: any move from a white square takes the robot to a black square, and vice versa. This makes it difficult for the robot to orient itself in this environment.

Finally, in the "Crossword Puzzle" environment, the robot is on a crossword puzzle grid such as the one in Figure 12. The robot has three actions available to it: it can step ahead one square, or it can turn left or right by 90 degrees. The robot can only occupy the white squares of the crossword puzzle; an attempt to move onto a black square is a "no-op." Attempting to step beyond the boundaries of the puzzle is also a no-op. Each of the four "walls" of the puzzle has been painted a different color. The robot looks as far ahead as possible in the direction it faces: if its view is obstructed by a black square, then it sees "black;" otherwise, it sees the color of the wall it is facing. Thus, the robot has five possible sensations. Since this environment is essentially a maze, it may contain regions which are difficult to reach or difficult to get out of.

In the current implementation, we have used an adaptive homing sequence or homing tree. We have also used the modified version of L^* described in Section 5-4.5. Finally, we have implemented a heuristic that attempts to focus effort on copies of L^* that have already made the most progress: if the homing sequence is executed and the L^* copy reached is not very far along, then the procedure is likely to re-execute the homing sequence to find one that is closer to completion. The idea of the heuristic is not to waste time on copies that have a long way to go. The heuristic seems to improve the running time for these three environments by as much as a factor of six.
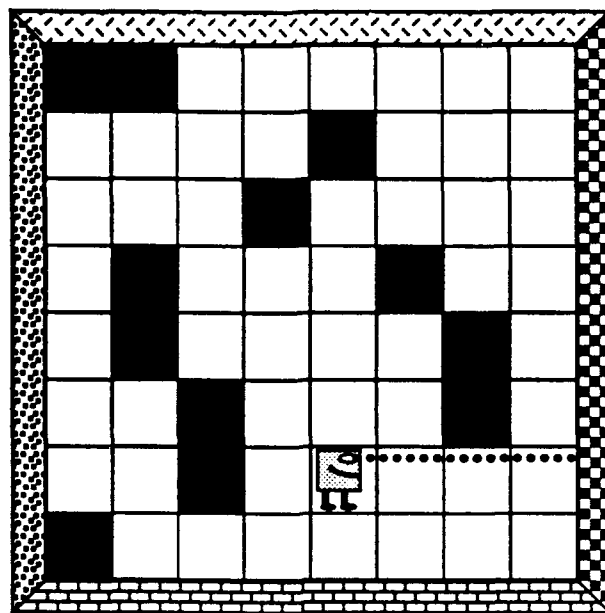
Figure 12: A crossword puzzle environment.

For the "Random Graph" and "Crossword Puzzle" environments, the inference procedure was provided in some experiments with an oracle which would return the shortest counterexample to an incorrect conjecture. All three environments were also tested with no external source of counterexamples; to find a counterexample, the robot would instead execute random actions until its model of the environment made an incorrect prediction of the output of some state.

Table 1 summarizes how our procedure handled each environment. In the table, "Source" refers to the robot's source of counterexamples: "S" indicates that the robot had access to the shortest counterexample, and "R" indicates that it had to rely on random walks. The column labeled "$|ran(\gamma)|$" gives the number of possible sensations which might be experienced by the robot. (Extending our algorithms to the case that the range of γ consists of more than two elements is trivial.) "Copies" is the number of copies of L^* which were active when a correct conjecture was made. "Queries" is the total number of membership and equivalence queries which were simulated. "Actions" is the total number of actions executed by the robot, and "Time" is elapsed cpu time in minutes and seconds. The procedure was implemented in C on a DEC MicroVax III. For example, inferring the 8×8 "Knight Moves" environment using randomly generated counterexamples required about 400,000 moves and 19 seconds of cpu time.

Note that for the "Random Graph" environment, the learning procedure sometimes did better with randomly generated counterexamples than with an oracle providing the shortest counterexample. It is not clear why this is so, although it seems plausible that in some way

Environment	size	n	k	|ran(γ)|	Source	Copies	Queries	Actions	Time
Random	25	25	3	2	S	20	1,108	10,504	:01.0
Graph					R	21	1,670	17,901	:01.2
	50	50	3	2	S	37	5,251	69,861	:06.0
					R	33	4,581	61,325	:03.6
	100	100	3	2	S	68	14,788	279,276	:24.1
					R	64	17,221	342,450	:18.1
	200	200	3	2	S	137	34,182	1,100,244	1:31.9
					R	136	29,796	1,012,279	:47.5
	400	400	3	2	S	275	72,027	3,010,377	4:52.0
					R	258	33,388	1,757,720	1:19.5
Knight	4	16	8	2	R	10	2,082	19,621	:01.4
Moves	8	64	8	2	R	50	17,818	385,678	:19.4
	12	144	8	2	R	88	22,208	780,595	:36.3
	16	256	8	2	R	124	63,476	3,855,520	2:41.9
	20	400	8	2	R	157	129,407	8,329,257	5:58.9
Crossword	4	48	3	5	S	41	2,424	30,285	:02.5
Puzzle					R	41	2,817	55,749	:04.1
	8	208	3	5	S	97	18,523	839,087	:52.9
					R	104	16,643	1,049,466	:51.0
	12	416	3	5	S	188	68,793	5,564,299	5:15.6
					R	193	58,222	8,850,079	7:12.5

Table 1: Experimental results.

the random walk sequences give more information about the environment. For example, the counterexamples often become subsequences of the homing sequence, and it may be that random walk counterexamples make for better, more distinguishing homing sequences.

In sum, the running times given are quite fast, and the number of moves taken far less than allowed for by the theoretical worst-case bounds. Nevertheless, it is also true that the number of actions executed is still somewhat large, much too great to be practical for a real robot. There are probably many ways in which our algorithm might be improved — both in a theoretical sense, and in terms of heuristics which might improve the performance in practice. We leave these questions as open problems.

5-9   Conclusions and open questions

We have shown how to infer an unknown automaton, in the absence of a reset, by experimentation and with counterexamples. For the class of permutation automata, we have shown that the source of counterexamples is unnecessary. We have described polynomial-time algorithms which are both state-based and diversity-based.

As discussed in the introduction, these results represent only modest progress toward our

ultimate goal, the development of a robot capable of inferring a usable model of its real-world environment. It is not clear how to get there from where we are now. To begin with, we need algorithms that are even more efficient than the ones described here. Perhaps more importantly, we need techniques for handling more realistic environments. These would include environments exhibiting various kinds of randomness or uncertainty, and also environments with infinitely many states. In such cases, inference of a perfect model will almost certainly be out of the question. What then is the best we can hope for? What are the skills most needed for the robot to function in its environment, and how can those skills be learned?

Bibliography

[1] William Aiello and Milena Mihail. Learning the Fourier spectrum of probabilistic lists and trees. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1991.

[2] James A. Anderson and Edward Rosenfeld, editors. *Neurocomputing: Foundations of Research*. MIT Press, 1988.

[3] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[4] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, August 1980.

[5] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[6] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[7] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[8] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley, Computer Science Division, August 1989. To appear, *Journal of the Association for Computing Machinery*.

[9] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[10] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18(2):155–193, April 1979.

[11] Eric B. Baum. On learning a union of half spaces. *Journal of Complexity*, 6(1):67–101, March 1990.

[12] Patrick Billingsley. *Probability and Measure*. Wiley, second edition, 1986.

[13] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, April 1987.

[14] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[15] Raymond Board and Leonard Pitt. On the necessity of Occam algorithms. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 54–63, May 1990.

[16] Ravi B. Boppana. Amplification of probabilistic Boolean formulas. In *26th Annual Symposium on Foundations of Computer Science*, pages 20–29, October 1985.

[17] Ravi Babu Boppana. *Lower Bounds for Monotone Circuits and Formulas*. PhD thesis, Massachusetts Institute of Technology, 1986.

[18] Stéphane Boucheron and Jean Sallantin. Some remarks about space-complexity of learning, and circuit complexity of recognizing. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 125–138, August 1988.

[19] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[20] Thomas G. Dietterich. Machine learning. In Joseph F. Traub, Barbara J. Grosz, Butler W. Lampson, and Nils J. Nilsson, editors, *Annual Review of Computer Science*, volume 4, pages 255–306. Annual Reviews, 1990.

[21] Gary L. Drescher. Genetic AI — translating Piaget into Lisp. Technical Report 890, MIT Artificial Intelligence Laboratory, February 1986.

[22] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[23] R. M. Dudley. Central limit theorems for empirical measures. *The Annals of Probability*, 6(6):899–929, 1978.

[24] Miroslav Fiedler. Bounds for eigenvalues of doubly stochastic matrices. *Linear Algebra and its Applications*, 5(3):299–310, July 1972.

[25] Sally Floyd. Space-bounded learning and the Vapnik-Chervonenkis dimension. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 349–364, July 1989.

[26] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, August 1990.

[27] Merrick Furst, Jeffrey Jackson, and Sean Smith. Learning AC^0 functions sampled under mutually independent distributions. Technical Report CMU-CS-90-183, Carnegie Mellon University, School of Computer Science, October 1990.

[28] E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[29] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[30] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. In *31st Annual Symposium on Foundations of Computer Science*, pages 193–202, October 1990.

[31] Sally Ann Goldman. *Learning Binary Relations, Total Orders, and Read-Once Formulas*. PhD thesis, Massachusetts Institute of Technology, September 1990. Available as Technical Report MIT/LCS/TR-483, MIT Laboratory for Computer Science.

[32] G. Györgi and N. Tishby. Statistical theory of learning a rule. In *Proceedings of the STATPHYS-17 Workshop on Neural Networks and Spin Glasses*, 1989.

[33] Thomas R. Hancock. Identifying μ-formula decision trees with queries. Technical Report TR-16-90, Harvard University, Center for Research in Computing Technology, 1990.

[34] David Haussler. Space efficient learning algorithms. Technical Report UCSC-CRL-88-2, University of California Santa Cruz, Computer Research Laboratory, March 1988.

[35] David Haussler. Generalizing the PAC model for neural net and other learning applications. Technical Report UCSC-CRL-89-30, University of California Santa Cruz, Computer Research Laboratory, September 1989. To appear, *Information and Computation*.

[36] David Haussler. Generalizing the PAC model: Sample size bounds from metric dimension-based uniform convergence results. In *30th Annual Symposium on Foundations of Computer Science*, pages 40–45, October 1989.

[37] David Haussler. Decision theoretic generalizations of the PAC learning model. Unpublished manuscript, 1990.

[38] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55, August 1988. Available as Technical Report UCSC-CRL-88-06, University of California Santa Cruz, Computer Research Laboratory. To appear, *Information and Computation*.

[39] David Haussler, Nick Littlestone, and Manfred K. Warmuth. Expected mistake bounds for on-line learning algorithms. Unpublished manuscript, April 1987.

[40] David Haussler, Nick Littlestone, and Manfred K. Warmuth. Predicting $\{0, 1\}$-functions on randomly drawn points. In *29th Annual Symposium on Foundations of Computer Science*, pages 100–109, October 1988.

[41] Lisa Hellerstein. *On Characterizing and Learning Some Classes of Read-Once Formulas*. PhD thesis, University of California at Berkeley, 1989.

[42] Lisa Hellerstein and Marek Karpinski. Read-once formulas over different bases. Unpublished manuscript, December 1990.

[43] David Helmbold, Robert Sloan, and Manfred K. Warmuth. Learning nested differences of intersection-closed concept classes. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 41–56, July 1989.

[44] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[45] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[46] John H. Holland. Genetic algorithms and adaptation. In Oliver G. Selfridge, Edwina L. Rissland, and Michael A. Arbib, editors, *Adaptive Control of Ill-defined Systems*. Plenum Press, 1984.

[47] Abraham Kandel. *Fuzzy Techniques in Pattern Recognition*. Wiley, 1982.

[48] Michael Kearns. Thoughts on hypothesis boosting. Unpublished manuscript, December 1988.

[49] Michael Kearns. *The Computational Complexity of Machine Learning*. MIT Press, 1990.

[50] Michael Kearns and Ming Li. Learning in the presence of malicious errors. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 267–280, May 1988.

[51] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of Boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 285–295, May 1987.

[52] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 433–444, May 1989.

[53] Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *31st Annual Symposium on Foundations of Computer Science*, pages 382–391, October 1990.

[54] Yves Kodratoff and Ryszard Michalski, editors. *Machine Learning: An Artificial Intelligence Approach*, volume III. Morgan Kaufmann, 1990.

[55] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

[56] Benjamin J. Kuipers and Yung-Tai Byun. A robust, qualitative approach to a spatial learning mobile robot. In *SPIE Advances in Intelligent Robotics Systems*, November 1988.

[57] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. In *30th Annual Symposium on Foundations of Computer Science*, pages 574–579, October 1989.

[58] Nathan Linial, Yishay Mansour, and Ronald L. Rivest. Results on learnability and the Vapnik-Chervonenkis dimension. In *29th Annual Symposium on Foundations of Computer Science*, pages 120–129, October 1988.

[59] Nick Littlestone and Manfred Warmuth. Relating data compression and learnability. Unpublished manuscript, November 1987.

[60] Yishay Mansour. Learning via Fourier transform. Unpublished manuscript, April 1990.

[61] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Master's thesis, Massachusetts Institute of Technology, May 1990. Technical Report AI-TR 1228, MIT Artificial Intelligence Laboratory.

[62] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, 1983.

[63] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann, 1986.

[64] Tom Mitchell, Bruce Buchanan, Gerald DeJong, Thomas Dietterich, Paul Rosenbloom, and Alex Waibel. Machine learning. In Joseph F. Traub, Barbara J. Grosz, Butler W. Lampson, and Nils J. Nilsson, editors, *Annual Review of Computer Science*, volume 4, pages 417–433. Annual Reviews, 1990.

[65] Tom M. Mitchell, Jaime G. Carbonell, and Ryszard S. Michalski, editors. *Machine Learning: A Guide to Current Research.* Kluwer Academic Publishers, 1986.

[66] Giulia Pagallo and David Haussler. A greedy method for learning μDNF functions under the uniform distribution. Technical Report UCSC-CRL-89-12, University of California Santa Cruz, Computer Research Laboratory, June 1989.

[67] Leonard Pitt. Inductive inference, DFAs, and computational complexity. Technical Report UIUCDCS-R-89-1530, University of Illinois at Urbana-Champaign, Department of Computer Science, July 1989.

[68] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, October 1988.

[69] Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, May 1989. Available as Technical Report UIUCDCS-R-89-1499, University of Illinois at Urbana-Ch ipaign, Department of Computer Science. To appear, *Journal of the Association for Co:..puting Machinery.*

[70] Leonard Pitt and Manfred K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, December 1990.

[71] David Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, 1984.

[72] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[73] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *28th Annual Symposium on Foundations of Computer Science*, pages 78–87, October 1987.

[74] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 411–420, May 1989.

[75] Ronald L. Rivest and Robert E. Schapire. A new approach to unsupervised learning in deterministic environments. In Yves Kodratoff and Ryszard Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume III, pages 670–684. Morgan Kaufmann, 1990.

[76] David E. Rumelhart and James L. McClelland, editors. *Parallel Distributed Processing*. MIT Press, 1986.

[77] Robert E. Schapire. Pattern languages are not learnable. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 122–129, August 1990.

[78] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[79] Robert Elias Schapire. Diversity-based inference of finite automata. Master's thesis, Massachusetts Institute of Technology, May 1988. Supervised by Ronald L. Rivest. Technical Report MIT/LCS/TR-413, MIT Laboratory for Computer Science.

[80] Jude W. Shavlik and Thomas G. Dietterich, editors. *Readings in Machine Learning*. Morgan Kaufmann, 1990.

[81] Robert H. Sloan. Types of noise in data for concept learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 91–96, August 1988.

[82] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the Association for Computing Machinery*, 22(2):215–225, April 1975.

[83] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[84] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 1982.

[85] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, XVI(2):264–280, 1971.

[86] Karsten Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–326, August 1990.

[87] Stuart W. Wilson. Knowledge growth in an artificial animal. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 16–23, July 1985.

[88] Kenji Yamanishi. A learning criterion for stochastic rules. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, August 1990.

[89] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, June 1965.