AD-A244 108

ENTATION PAGE

Form Approved
OMB No. 0704-0188

imated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, id reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this is burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson he Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

PORT DATE March 1991	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
On the Errors that Learning Machines Will Make

5. FUNDING NUMBERS
DAAL03-88-K-0082
(2)

6. AUTHOR(S)
A.W.Biermann, K.C.Gilbert, A.Fahmy, B.Koster

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Dept.
Duke University
Durham, NC 27706

DTIC
ELECTE
JAN 10 1992
S D
D

8. PERFORMING ORGANIZATION REPORT NUMBER
CS-1991-32

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 25795.10-MA-AI

11. SUPPLEMENTARY NOTES
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Associated with each learning system there is a class of learnable behaviors. If the target behavior to be acquired is in the learnable class, it will be learned perfectly. If it is outside that class, the machine will only be able to acquire a behavior that approximates the target and it will always make errors. It is desirable for a learning machine to have a large learnable class to maximize the chances of acquiring the unknown behavior and to minimize the expected error when only an approximation is possible. However, it is also desirable to have a small learnable class so that learning can be achieved rapidly. Thus the design of learning machines involves selecting a position on the spectrum: minimum error and slow learning time versus larger error and faster learning time.

Machines that have fast learning times, relatively small learnable classes, and thus relatively large expected errors are called "realization sparse" in this paper. It is shown that many common learning systems are of this type including signature tables, linear system models, and conjunctive normal form expression based systems.

These studies lead to the concept of an "optimum" machine which spreads its learnable behaviors across the behavior space in a manner to minimize the expected error. An approximation to such optimum machines is presented and its behavior is compared to the more traditional learning machines.

14. SUBJECT TERMS
Learning machines, learnable classes

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

CS-1991-32

On the Errors that Learning Machines Will Make

A.W.Biermann, K.C.Gilbert,
A.Fahmy, B. Koster

Department of Computer Science

Duke University

Durham, North Carolina 27706

On The Errors That Learning Machines Will Make

A.W. Biermann, K.C. Gilbert, A. Fahmy, B. Koster

Department of Computer Science
Duke University
Durham, NC 27706

Revised March 1991

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and/or Special
A-1

DTIC COPY INSPECTED

ABSTRACT

Associated with each learning system there is a class of learnable behaviors. If the target behavior to be acquired is in the learnable class, it will be learned perfectly. If it is outside that class, the machine will only be able to acquire a behavior that approximates the target and it will always make errors. It is desirable for a learning machine to have a large learnable class to maximize the chances of acquiring the unknown behavior and to minimize the expected error when only an approximation is possible. However, it is also desirable to have a small learnable class so that learning can be achieved rapidly. Thus the design of learning machines involves selecting a position on the spectrum: minimum error and slow learning time versus larger error and faster learning time.

Machines that have fast learning times, relatively small learnable classes, and thus relatively large expected errors are called "realization sparse" in this paper. It is shown that many common learning systems are of this type including signature tables, linear system models, and conjunctive normal form expression based systems.

These studies lead to the concept of an "optimum" machine which spreads its learnable behaviors across the behavior space in a manner to minimize the expected error. An approximation to such optimum machines is presented and its behavior is compared to the more traditional learning machines.

This paper is based on work supported by the U.S. Army Research Office under Grant DAAG-29-84-K-0072, Grant DAAL03-88-K-0082 and the Air Force Office of Scientific Research Grant No. 81-0221.

92-00842

Table of Contents

Introduction	2
Three Example Learning Machines and Some Questions	4
Relating the Number of Learnable Behaviors to Error and Time to Learn	12
Signature Tables	22
Linear Models	27
Conjunctive Normal Form Expressions	29
In Search of the Optimal Learning Machine: Truncation Machines	30
In Search of the Optimal Learning Machine: The G1-Machine	33
The Random Vectors Machine	41
Comparing Some Expected Errors	42
Conclusions	45
Appendix A: Counting Signature Tables	47
Appendix B: The Average Error for Truncation Machines	53
References	56

Introduction

A basic characteristic of many learning systems is that they are able to acquire only a limited class of behaviors, a class that may be far short of the set of all possible behaviors. If the target behavior is in the learnable class, the learning machine may be able to acquire it as training information becomes available. If the target behavior is outside of the learnable class, it will not be achieved. The best the system can do, in this case, is to try to find a learnable behavior close enough to the target to give few errors and satisfactory if not perfect behavior.

Learnability classes may be studied in terms of basic characterizations and various dimensions including their sizes, the amount of information or time required to learn, and the distribution of the class across the space of all possible behaviors. It is desirable to have a large learnable class because this increases the chances of learning an unknown function or of approximating it accurately. However, if the error problems are not too severe, it is advantageous to have a small learnable class because then convergence to the final solution is much faster. Thus there is a necessary tradeoff between accuracy and rate of learning which the system designer must understand. Another issue of interest is the distribution of the learnable class in the space of all possible behaviors. If the class is uniformly spread across the space, randomly selected behaviors within the space will tend to all be learned equally well. However, if the class is clustered in one or a few regions, target behaviors within those regions will be acquired well and others will be acquired poorly or not at all.

This paper is concerned with these issues. First we examine relationships between the number of learnable functions, the time required to learn, and the worst case and expected rates of errors. Next we define a class of learning machines which are called *realization sparse* machines and show that large such machines will do only slightly better than a random coin flipping machine on most learning problems. Then we examine several common types of

machines, specifically the signature table systems, linear models, and conjunctive normal form machines and show that they are all realization sparse. In fact, we show that all learning machines that learn within a reasonable time are realization sparse. These studies lead to the concept of an optimal or minimum expected error learning machine and an approximation to such machines is developed. Comparisons of known machines to optimal or near optimal machines are made to obtain a measure of their quality.

In order to keep the complexity of the study within bounds, the assumed model is a binary function learner as shown in Figure 1 with p binary inputs. The various learning systems to be studied are all adapted to acquire functions of this form so that their characteristics become comparable. This paper will ignore the details of learning algorithms presented in the literature and concentrate on examining the learnable classes and their characteristics.

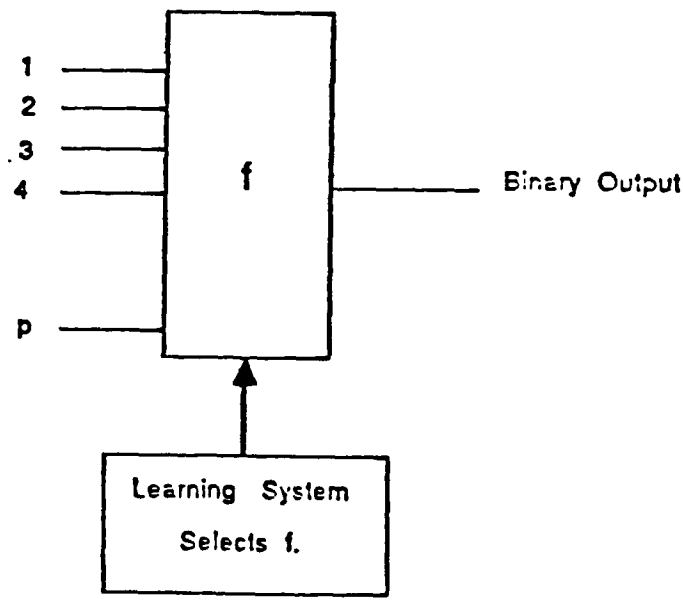


Figure 1. The learning system model

The next section will present three example learning machines and formulate more precisely the questions to be examined in this paper. Then a section will examine the nature of the

tradeoff between the expected error for a system and the time or amount of information required to learn. The following sections will overview some results on three learning models. Then the possibility of building a perfect minimal expected error learning machine will be examined. Such an ideal machine has interest in its own right and serves as a comparison for other more easily realized machines. Finally, a few summary comments conclude the paper.

Three Example Learning Machines and Some Questions

This study begins with a cursory examination of three learning machines, a signature table system, a linear system, and a conjunctive normal form machine. All will follow the model of Figure 1 with $p = 4$.

The example signature table system appears in Figure 2. The constants c_w may only have values 0 or 1 and the output is computed by indexing through the tables starting with the inputs at the bottom. That is, the values of the x_i's determine the output values of the lower two tables, and these values become the inputs to the top table which yields the system output. For example, an input of $(x_1, x_2, x_3, x_4) = (0,1,1,0)$ yields outputs of $c_{0,1}=1$ and $c_{1,2}=0$ from the lower two tables. These become the input 1,0 for the top table which gives an output of 1. Similarly, all of the sixteen possible values for the vector (x_1, x_2, x_3, x_4) can be input to this system resulting in the total system behavior shown in Figure 5. Learning is done by varying the values of the c_w's and Samuel [14] and Biermann *et al.* [1] have given algorithms for doing this type of learning.

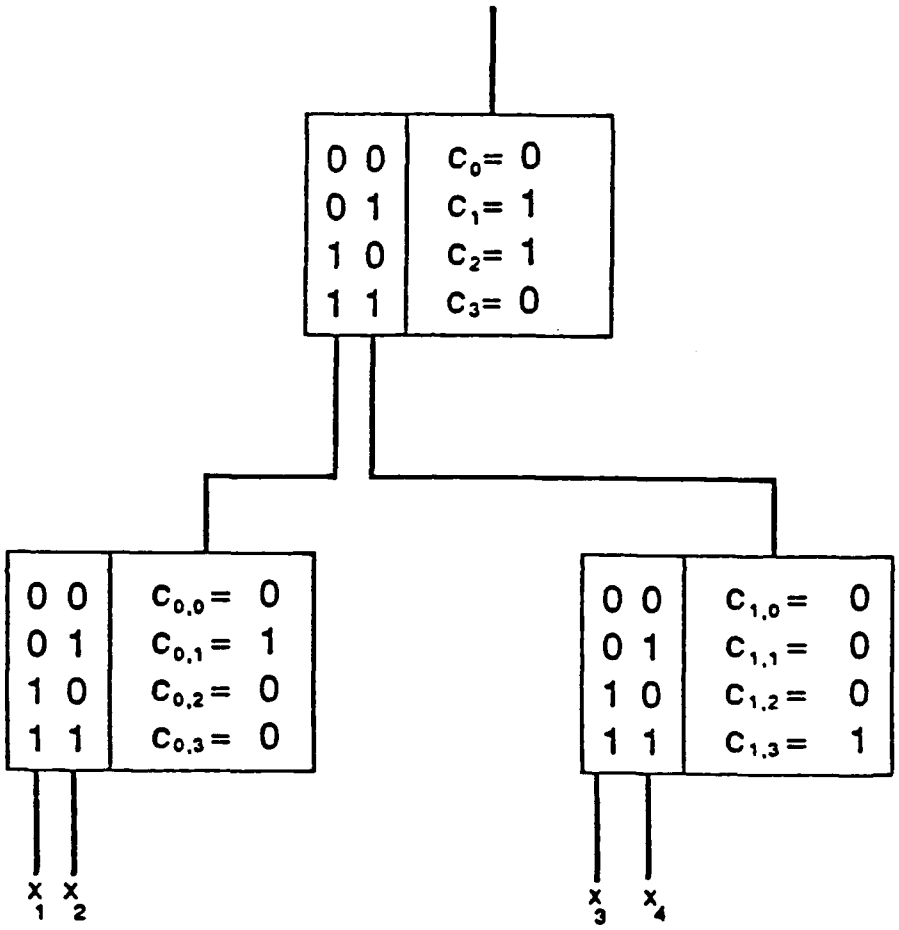
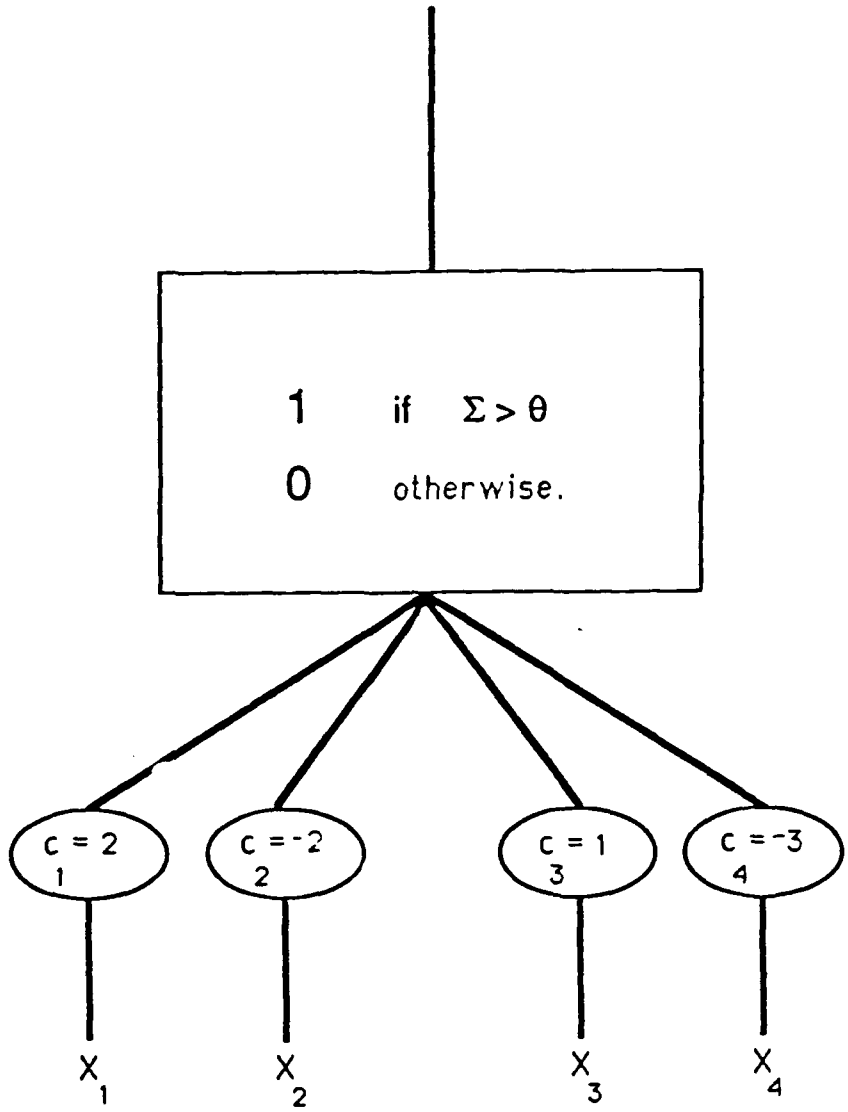


Figure 2. A signature table system.

An example linear system appears in Figure 3. Here the coefficients c_i and θ can take on real values and the function output is 1 if $\sum_{i=1}^{4} c_i x_i > \theta$ and 0 otherwise. Thus for the coefficient values given, the input vector $(x_1, x_2, x_3, x_4) = (0,1,1,0)$ gives a value for $\sum_{i=1}^{4} c_i x_i$ of -1 and a function output of 0. The total function behavior for this machine is shown in Figure 5. Again, learning is achieved by varying the coefficients c_i and θ. Some algorithms for learning are given by Minsky and Papert [8], Nilsson [10], and Samuel [13].



A linear system.

Figure 3.

A third learning system is illustrated in Figure 4, a Boolean conjunctive normal form acquisition machine. Here the input variables and/or their complements are combined into every possible Boolean sum of k items or fewer (where $k=2$ in this example). Then the sums with associated coefficients c_i which equal 1 are combined into a Boolean product to produce

the output. If we assume all coefficients are zero in Figure 4 except for c_2, c_3, and c_4, then the example system computes the function $(x_1+x_3)x_2'(x_1'+x_4)$. Thus the example input $(x_1,x_2,x_3,x_4)=(0,1,1,0)$ will yield $(0+1)1'(0'+0)=0$. All other inputs can be similarly tabulated as shown in Figure 5. Learning is done by adjusting the coefficients c_i which each may have the value of 0 or 1 and a learning algorithm is given by Valiant [16].

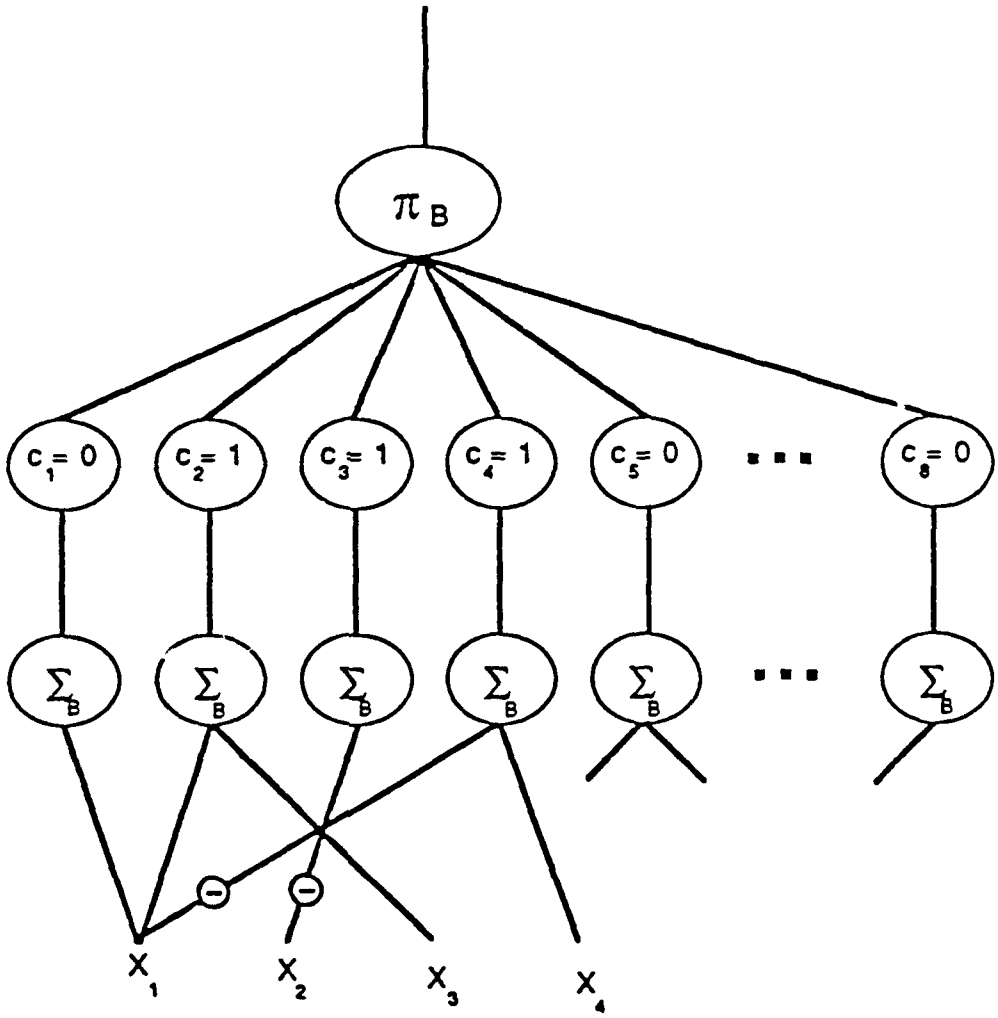


Figure 4. A Boolean conjunctive normal form machine.

$x_1x_2x_3x_4$	Signature Table (Fig 2)	Linear System (Fig 3)	Conjunctive Normal Form (Fig 4)
0000	0	0	0
0001	0	0	0
0010	0	1	1
0011	1	0	1
0100	1	0	0
0101	1	0	0
0110	1	0	0
0111	0	0	0
1000	0	1	0
1001	0	0	1
1010	0	1	0
1011	1	0	1
1100	0	0	0
1101	0	0	0
1110	0	1	0
1111	1	0	0

Figure 5. The learnable vectors for the above three machines.

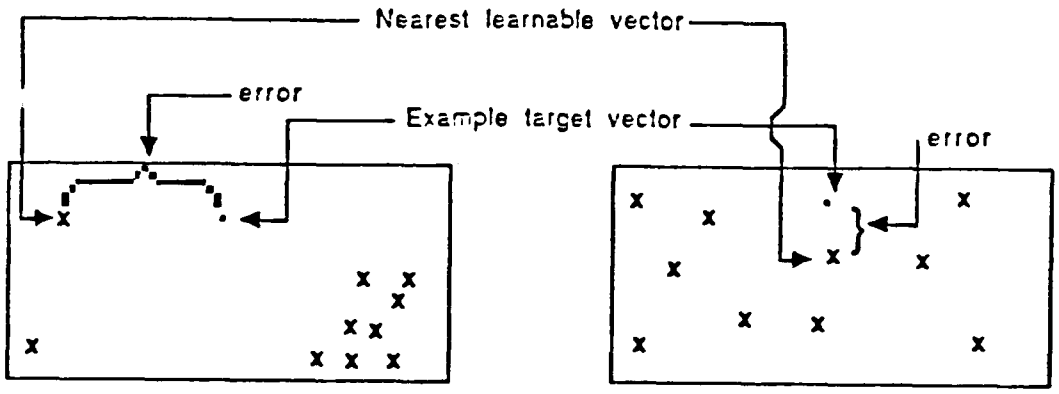
Figure 5 thus shows, for each of three kinds of learning machines, a learnable behavior. The 16 bit vectors which are columns in this table are called *learnable vectors*. Each type of machine has a set of possible learnable vectors which can be achieved by varying its constants. These sets of learnable vectors are called *learnable classes* and we will use the symbol L to designate the size of such a set for a given machine. Usually L falls far short of the total number of possible such vectors.

When the learnable vectors do not cover the whole space, the learning machine may be asked to acquire a behavior outside its class. In this case, it can select a vector with minimum Hamming distance to that target vector. It may have difficulty finding a learnable vector close to the target if its learnable vectors are clustered as shown in Figure 6(a). On the other hand, the learnable vectors may be spread across the space uniformly as in Figure 6(b) so that every possible target vector will be near a learnable vector. If the learnable class is spread so as to minimize the expected Hamming distance [4] from a randomly selected target vector to the closest learnable vector, then the learning machine will be called *optimum*.

The following questions will be examined here:

1. What are the general relationships for all learning machines between the number of learnable functions, the number of examples required to learn, and the rates of errors.
2. What is the nature of the learnable class for each type of machine?
3. If a target behavior is selected randomly from the space of all possible behaviors, what is the probability that the learning machine will be able to acquire that behavior?
4. If a target behavior is not learnable but the machine adapts to give the closest possible behavior, what will be the expected number of errors?
5. What is the nature of an optimal learning machine which has its learnable class spread across the behavior space to minimize expected error?

6. How do the known learning machines compare with this optimal machine?



a. A clustered distribution with high expected error.

b. A uniformly distributed class with minimum expected error.

Figure 6. Two possible distributions for $L = 10$ learnable vectors (designated by λ^{τ} s) in a behavior space.

Relating the Number of Learnable Behaviors to Error and Time to Learn

In general, as the number L of learnable vectors increases, the expected distance from a randomly chosen vector to its nearest learnable vector decreases and the expected time to learn increases. These relationships are examined in this section. Specifically, a bounding formula is found showing the relationship between the number of examples required to learn and the worst case error that must occur. Second, it is shown that the expected error is near the worst case error so that these worst case results cannot be taken lightly. Worst case behavior is not

obscure behavior, it is typical. Finally, an experimental simulation is described that shows how the time required to learn is related to L.

Relating L with Worst Case Error. Let S_0 be the set of vectors which are as near or nearer to a given learnable vector v_0 as to any other learnable vector. That is, S_0 is the set of possible target vectors that would result in the selection of v_0 as the most desireable learned behavior. We will say that S_0 is the set of vectors *covered by* v_0. If the worst case error in S_0 is D (i.e., the vector in S_0 with greatest Hamming distance from v_0 differs from v_0 in D positions), then the number of vectors in S_0 is no more than $\sum_{i=0}^{D} \binom{n}{i}$ where n is the vector length. (There are $\binom{n}{i}$ vectors at distance i from v_0 for each $i = 0,1,2,\ldots,D$.)

Suppose all of the L learnable vectors cover sets with worst case error D, then the set of vectors covered by them all is no greater than $L \sum_{i=0}^{D} \binom{n}{i}$. But the total number of such vectors is 2^n so

$$L \sum_{i=0}^{D} \binom{n}{i} \geq 2^n \quad . \tag{1}$$

This inequality relates L and D and often provides useful information. For example, if in an application someone has specified a maximum allowed worst case error D, then the parameters of the learning machine must be adjusted to make L large enough. If the learnable vectors of the learning machine are widely dispersed as in Figure 6b, then (1) will give a good estimate of the required value of L. If the learnable vectors are clustered (Figure 6(a)), then (1) will give only a distant lower bound for the required value of L. Also it is clear that at least $\log_2 L$ observations must be made in order to differentiate the final learned behavior from the L alternatives. So (from (1) above) the number of observed input-output observations in order to guarantee a learned behavior with no more than D errors is no lower than $n - \log_2 \sum_{i=0}^{D} \binom{n}{i}$.

Relationship (1) can be put into more understandable form using an approximation given by Hamming [4], page 165 . If we define λ to be $\lambda = D/n$, then under the condition

$$n \geq \frac{1-\lambda}{2\pi\lambda\,(1-2\lambda)^2}$$

the following relationship holds:

$$\sum_{i=o}^{\lambda n} \binom{n}{i} \leq 2^{nH(\lambda)} \qquad (2)$$

where $H(\lambda) = -\log\,(\,\lambda^{\lambda}(1-\lambda)^{1-\lambda})$. Substituting (2) into (1) and rearranging yields a form whose trends are clear for large n.

$$\frac{\log_2 L}{n} \geq 1 - H(\lambda) \qquad (3)$$

If the allowed percentage of errors λ is to be small, then $H(\lambda)$ will be small also. This means that the number $\log_2 L$ of observations needed to learn needs to be nearly as large as n, the total number of values of the function.

As an example, suppose a designer is building a game playing program with 30 3-bit features and requires learning to achieve 95 percent accuracy. Then the learning machine will require at least

$$2^{90}\,(1 - 0.28) = 0.72 \times 2^{90}$$

examples to guarantee that the required accuracy will be met. In other words, the learning machine will have to observe 72 per cent of all function values to achieve the specified accuracy.

Suppose we have a class of learning machines, such as the signature tables or perceptrons, with the property that there is a member of the class for each integer p. Then one can define L_p to be the number of learnable functions for the member of the class with p inputs. Then (3) can be rewritten as

$$\frac{\log_2(L_p)}{2^p} \geq 1 - H(\lambda) \qquad (4)$$

We will say a class of learning machines is *realization sparse* if the left side of (4) approaches zero as p becomes large.

The minimum number $\log_2(L_p)$ of observations required to learn, as specified by (4), varies as shown in Figure 7. This curve shows the tradeoff between low error with many learning examples and high error with few. This relationship is a fundamental law for all machines that match the model. Realization sparse classes have the property that $\log_2(L_p)$ is very small compared to 2^p if p is large and this leads to worst case errors near 50 percent. It is a basic result of this paper, that many if not most of the commonly studied learning machines are realization sparse.

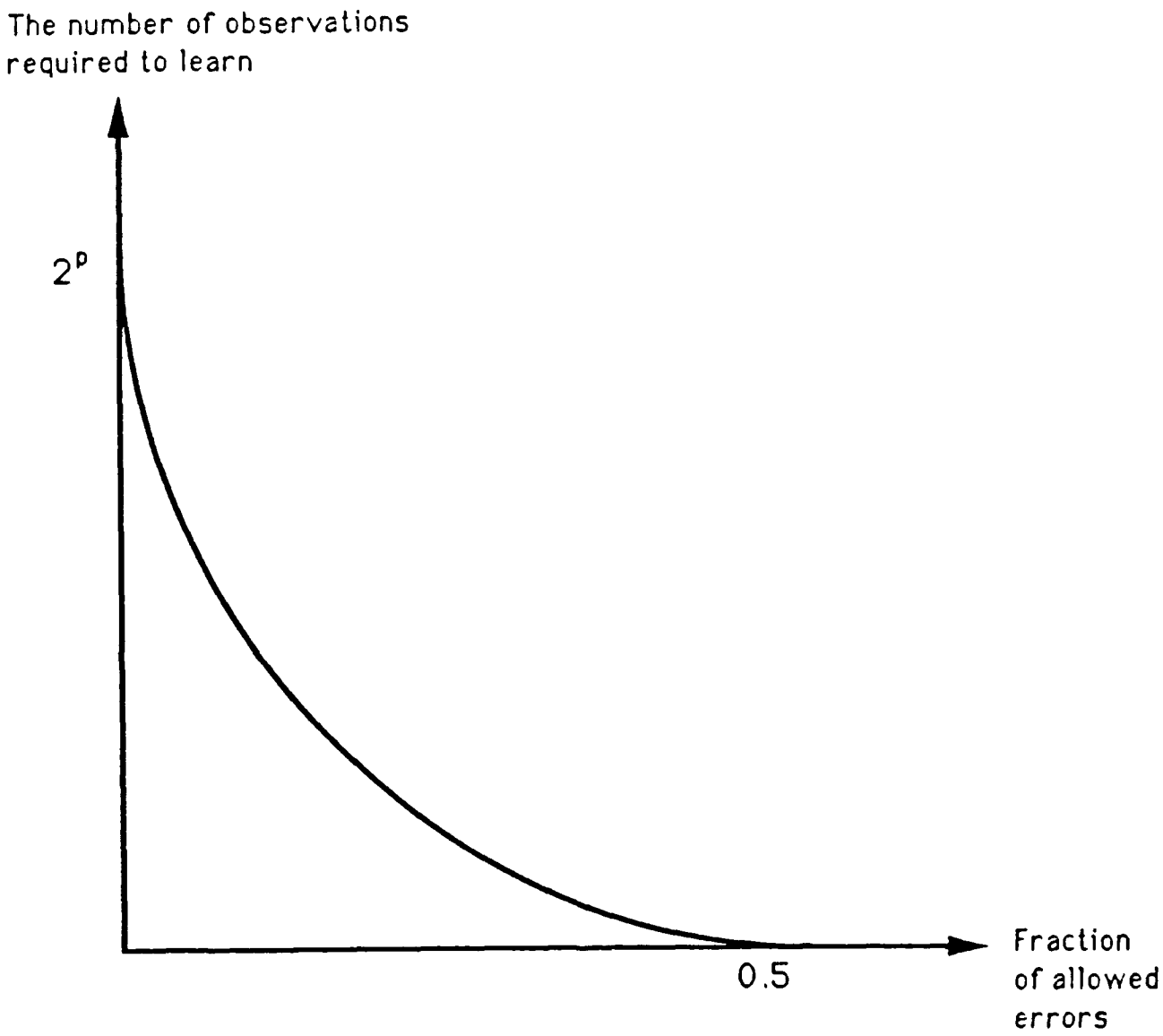


Figure 7. Minimum bound on the number of observations required to learn versus the fraction λ of allowed errors.

Relating L with Expected Error. The above observations relate to the worst case error that could occur if the target function is chosen randomly from the space of all possible functions. One can hope that the expected error will be considerably less. In fact, the expected error will

be greater than two thirds the worst case error D as can be seen from the following argument.

In order to get a lower bound on the expected error, assume the n-space is divided into L spheres that surround learnable vectors where each sphere has diameter D. A less uniform distribution of learnable vectors would yield a higher expected error so this assumption is acceptable when computing a lower bound. There will be $\binom{n}{i}$ vectors at radius i from each v_0 for each i. This number can be represented for large n by the normal distribution as shown in Figure 8. It is clear that the expected value for a normal distribution is greater than that for, say, the linear approximation shown which has an expected value of $\frac{2}{3}D$. (We assume that D is low enough to be below the inflexion point on the normal curve:

$$D < \frac{1}{2}\left(n - \sqrt{\frac{n}{2}}\right) .)$$

This is the desired result; the expected value for error is 2/3 as large as the worst case examined above, so the graph of Figure 7 gives a good approximation to typical results as well as worst case results.

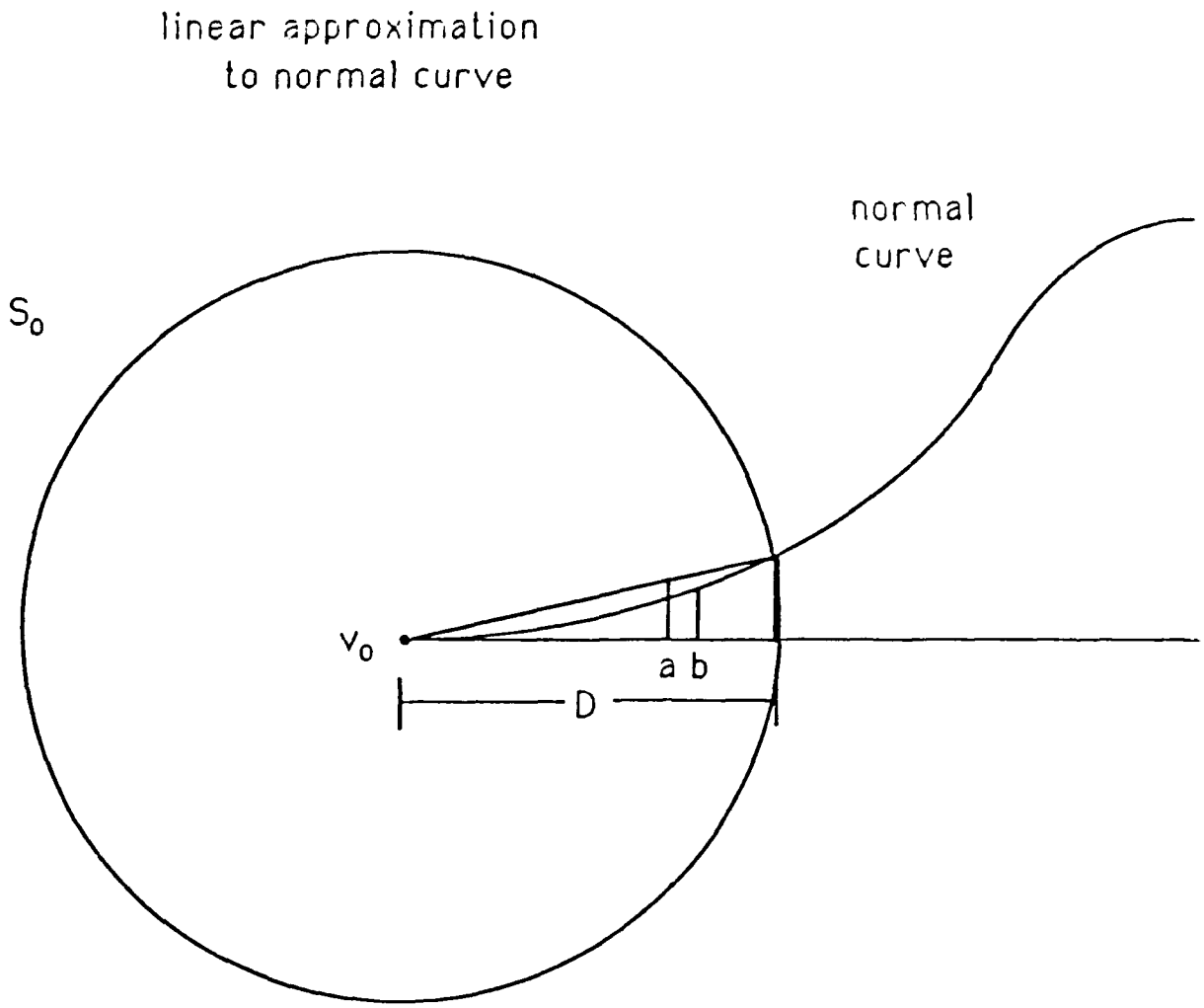


Figure 8. The expected error b will be greater than $a = 2/3D$.

A more accurate lower bound to expected error can be computed by accounting precisely for all of the 2^n vectors. If D is the worst case error for v_0, the sphere of vectors at distance D-1 and less from v_0 will be completely full, and as few as possible will be at distance D from

v_0. Remember that this means that for a given L, D is the lowest integer such that (1) holds. With this assumption (which is not realizable for every L), one can compute the expected error. It provides a lower bound for any realizable expected error.

$$expected\ error\ lower\ bound = \frac{\sum_{i=0}^{D} i \times (number\ of\ vectors\ at\ distance\ i)}{2^n}$$

$$= \frac{\sum_{i=0}^{D-1} i \times (number\ of\ vectors\ at\ distance\ i) + D \times (number\ of\ vectors\ at\ distance\ D)}{2^n}$$

$$= \frac{\sum_{i=0}^{D-1} i \times L\binom{n}{i} + D \times (2^n - (number\ of\ vectors\ below\ distance\ D))}{2^n}$$

$$= \frac{L\sum_{i=0}^{D-1} i\binom{n}{i} + D\,(2^n - L\sum_{i=0}^{D-1}\binom{n}{i})}{2^n} \tag{5}$$

This lower bound is used as a comparison figure for various machines later in the paper.

The relationship (1) is often mentioned in studies of coding theory where the goal is to select codes (vectors) which are maximally distant from each other in order to maximize correctability. (See [4,11]) Incoming vectors which are near a legal code (using Hamming distance) are assumed to be erroneous versions of that code and are corrected to it. An interesting study in coding theory relates to "perfect codes" in which relationship (1) holds as an equality.

$$L \sum_{i=0}^{D} \binom{n}{i} = 2^n$$

For example, a solution for this equation is provided by the Golay code ([11], page 70) where $n = 23$, $D = 3$, and $L = 2^{12}$ and the associated code is well known. In the current domain, this

corresponds to having 4096 learnable vectors of length 23 with every one of the 2^{23}=8,388,608 possible behaviors being within 3 or less of some learnable vector. Thus learning 12 bits of information (out of 23) is enough to reduce the probability of error to no more than 3/23 assuming all inputs are equally likely. (The reader might wish to ponder the seeming contradiction that learning 12 bits apparently results in "knowing" 20 bits.)

Relating L to Time to Learn. It is difficult to compute the time required to learn in comparison with L because it depends on the learning algorithm used, the particular behavior that is to be acquired, and the information available about the target behavior. Suppose the learnable vectors are $v_1, v_2, \ldots, v_L$ and that the target vector is v_T. Suppose further that learning is to be done by observing randomly selected input-output pairs. That is, random positions in v_T are observed during learning and the best learnable vector at any given time is selected knowing only these few entries in v_T. Let s denote the set of inputs for which the output has been observed and let $| v |_s$ denote vector v with all entries corresponding to inputs not in s set to zero. For example, if $p=2$, v_T is as shown,

input	v_T
0 0	1
0 1	0
1 0	1
1 1	1

and it has been observed during learning that 0 0 yields 1 and 1 0 yields 1, then $s=\{00,10\}$ and $| v_T |_s=(1,0,1,0)$. One could propose that the learning algorithm should find the first v_i in the learnable class such that the Hamming distance between $| v_T |_s$ and $| v_i |_s$ is minimized. If we agree that v_T is *learned* when this learning algorithm selects a learnable vector and never again changes its guess, then Figure 9 shows for each time t the probability that v_T is learned as a function of L for $n=4$. Figure 9 assumes that an optimum set of L vectors is to be used

(where "optimum" is defined in the previous section). It indicates the way the rate of learning varies with L for such an optimum machine and other algorithms will be no faster. The expected decrease in learning speed is observed as L increases.

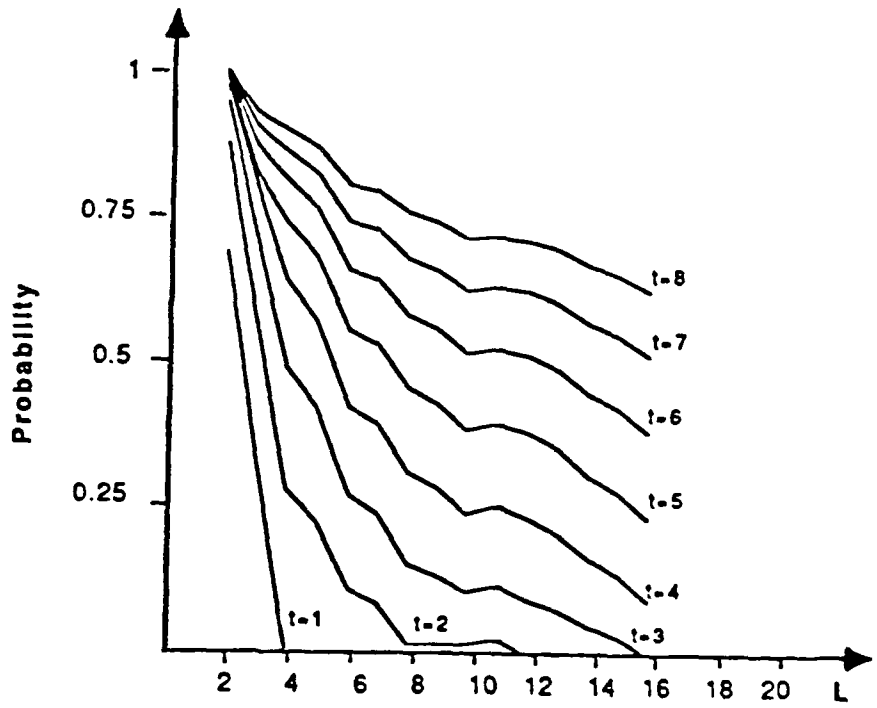


Figure 9. Probability that learning is complete versus

L for $n = 4$.

Signature Tables

One of the goals of the paper is to examine properties common to all learning systems as was done in the previous section. Another goal is to look at properties of specific learning systems such as signature tables, linear systems, and conjunctive normal form systems. The section begins the latter task by examining the characteristics of signature tables. Specifically, we would like to know

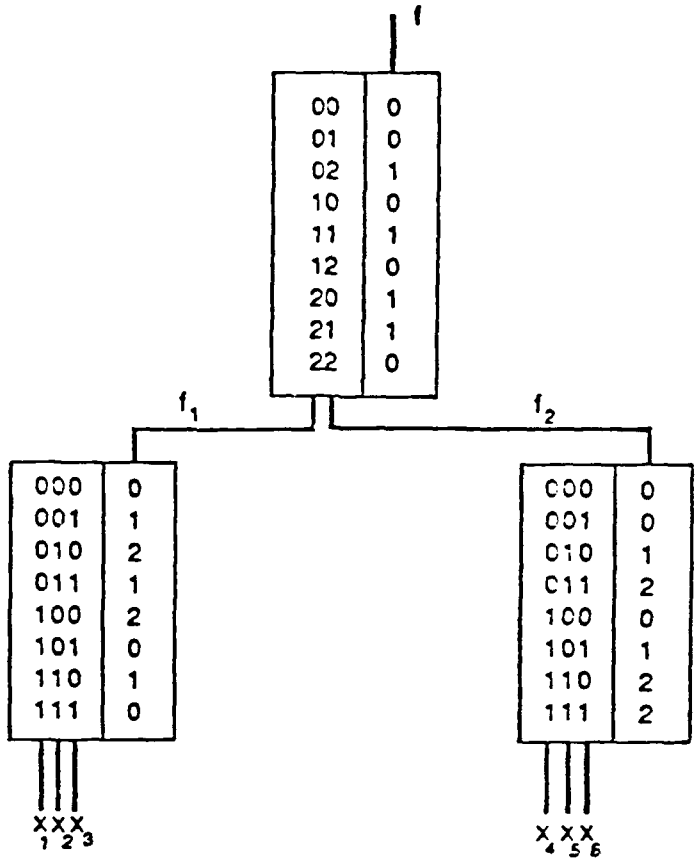
(a) what is the nature of the learnable vectors for signature table systems, and

(b) how many learnable vectors are there for given table configurations.

The answer to (b) will make it possible to find the probability that an unknown target behavior will be learnable by such systems. It will also give a way of estimating worst case error using (1) from the previous section.

The characterization of the learnable functions. The nature of the learnable vector will be studied for the type of signature table system T as shown in Figure 10. This system is best understood by constructing the matrix M(f,S) where f is the function that is computed by T and S is the subset of the inputs to T that feed into a single table in T (Biermann *et al.* [1]). M(f,S) is built by constructing one row for every possible value of the inputs S and one column for every possible value of the inputs not in S. The value of an entry in row i and column j of M(f,S) is the value of f when f receives input values in S associated with row i and the input values not in S associated with column j. Thus in the case of Figure 10, the value of f corresponding to input $(X_1,X_2,X_3,X_4,X_5,X_6) = (0,0,1,0,1,0)$ is 1. Let $S = \{X_1,X_2,X_3\}$ and construct M(f,S). Then the entry in the row labelled $(X_1,X_2,X_3) = (0,0,1)$ and the column labelled $(X_1,X_5,X_6) = (0,1,0)$ should be 1. Similarly, all other entries in M(f,S) can be constructed as shown in Figure 10.

T:



M(T) :

$X_1X_2X_3$	f_1 \ f_2	$X_4X_5X_6$ 000 0	001 0	010 1	011 2	100 0	101 1	110 2	111 2
000	0	0	0	0	1	0	0	1	1
001	1	0	0	1	0	0	1	0	0
010	2	1	1	1	0	1	1	0	0
011	1	0	0	1	0	0	1	0	0
100	2	1	1	1	0	1	1	0	0
101	0	0	0	0	1	0	0	1	1
110	1	0	0	1	0	0	1	0	0
111	0	0	0	0	1	0	0	1	1

Figure 10. An example table system T with its matrix M(T).

The theorem is that f can be realized by T if and only if for every S that comprises the inputs to a table in T, the size of the output alphabet in that table is at least as large as the number of distinct rows in M(f,S). In Figure 10, this means that since f_1, can take on only three values (0,1,2), $M(f\ ,\{x_1,x_2,x_3\})$ must have three or fewer distinct rows. Furthermore $M(f\ ,\{x_4,x_5,x_6\})$ can have only three or fewer rows because f_2 can have only three values. These limitations force a kind of repetitiveness on any realizable function f, and they precisely specify the set of functions that such a system can realize.

If the limitations on output alphabet size are relaxed, larger classes of functions can be realized but the tables quickly become very large. In the extreme case where there are no limitations on alphabet size, the table becomes simply an enumeration of every possible input and its associated output and it can realize any function.

The characterization holds for table systems of any depth. For example, consider Figure 11 and let v_i represent the alphabet size for f_i. Then the row multiplicity of $M(f\ ,\{x_1,x_2\})$ is v_3, of $M(f\ ,\{x_3,x_4\})$ is v_4, of $M(f\ ,\{x_1,x_2,x_3,x_4\})$ is v_1, and so forth. A function can be represented by a signature table system if and only if the kind of repetitiveness observed here occurs in f.

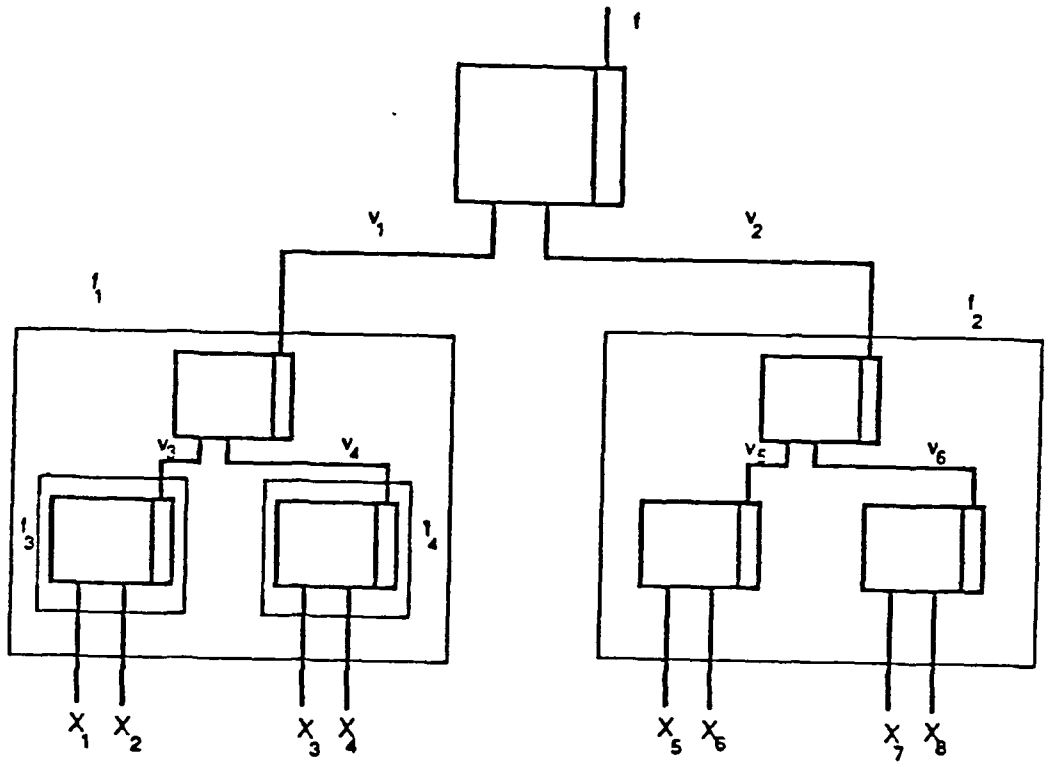


Figure 11. A three level signature table system.

Other properties of signature tables have been given by Biermann, *et al.* [1]. These results are related to studies on switching circuit decomposition as presented, for example, by Curtis [3].

Counting the number of learnable functions. A methodology for computing the number of functions realizable by a given signature table system is given by Biermann *et al.* [1] and an improved version appears as Appendix A in this paper. As an example, the number of functions realizable by table systems of the form in Figure 10 if f_1 and f_2 are binary can be computed for p inputs assuming $p_1 = p/2$ of the inputs are to the lower left table and $p_2 = p/2$ inputs are to the lower right table where p is even. If p is odd, then the two lower tables receive, respectively, $p_1 = \frac{p-1}{2}$ and $p_2 = \frac{p+1}{2}$ inputs. In fact, the number of functions is given by

$$L_p = 5 \cdot 2^{2^{p_1}+2^{p_2}-1} - 2^{2^{p_1}+2} - 2^{2^{p_2}+2} + 8 .$$

Returning to the definitions given above, it is easy to check that this class of machines is realization sparse. Therefore, as p increases, we can expect L_p to become small with respect to the number of functions 2^{2^p} and the worst case error to approach 50 percent. The following table gives an evaluation for these quantities and shows how quickly the trend becomes apparent.

Number of inputs p	Number of realizable functions L	Total number of functions	Lower bound on worst case error D	Number of function values	Percent error
1	4	4	0	2	0
2	16	16	0	4	0
3	88	256	1	8	12
4	520	65536	2	16	12
5	9160	4294967296	6	32	19
6	161800	1.844×10^{19}	15	64	23
7	41679880	3.403×10^{38}	34	128	27
8	10736893960	1.158×10^{77}	78	256	30

If deeper trees are examined as shown in Figure 11, for example, the trend is also worse. Consider the set of binary trees of depth d where d is the number of levels of tables. (Thus

d=3 in Figure 11.) Assume that all tables are restricted to vocabulary sizes of two or less. Then it is shown in Appendix A that the ratio of the number of realizable functions to possible functions is less than $1/(3(2^{2^{(2^d-1)}}))$ if d is greater than 2.

In conclusion, signature table systems can realize functions that have the kind of repetitiveness described here: The $M(f,S)$ matrices can have only as many distinct rows as there are alphabet symbols in the associated table. The counting results indicate that such table systems can realize relatively few functions. Most classes of signature table systems are realization sparse.

Linear Models

Linear models have been discussed extensively in the literature beginning with the studies of threshold functions (Muroga [9] and Nilsson [10]) and perceptrons, Minsky and Papert [8]), continuing with linear evaluation methodologies in applications such as game playing (Samuel [13]), and including more recently studies in connectionist learning (Rumelhart et al. [12], McClelland et al. [5] Sejnowski [15]). Characterizations of the learnable classes of functions have been given in terms of linear separability of points in a p-dimensional space (Muroga [9], Nilsson [10]) and in terms of computability of geometric properties on a two dimensional grid (Minsky and Papert [8]). A characterization has also been given in terms of function monotonicity as will be described here (from Muroga [9]).

Consider two inputs to a linear function f as described in the second section while all other inputs remain constant. Suppose these inputs take the values 00 and then 01 and that the output from f simultaneously goes from 0 to 1. Then one can conclude that the weight on the bit that changed is positive. Given that this weight is positive, one can be sure that on inputs 10 and 11, there cannot be a transition in the output from a 1 to a 0 since this would require a negative weight. A generalization of this idea results in the concept of a completely monotone

function: Consider for any function f the outputs $f(X_A)$ and $f(X_{\overline{A}})$ where the subscript A represents a particular setting of certain input bits in vector X and subscript $\overline{A}$ represents the opposite setting, with all other bits in vector X remaining the same. If for all input vectors X, the transition of outputs for the input transition X_A to $X_{\overline{A}}$ is in the same direction or makes no change, then the function is considered monotone with respect to A. If the function is monotone with respect to all A, then it is *completely monotone.*

The characterization as explained in Muroga [9] is that every linear function is completely monotone. Furthermore, every completely monotone function with $p \leq 8$ is realizable as a linear function. However, there exist completely monotone functions with p=9 which are not realizable.

Concerning the number L of realizable functions using the linear model, no formula has been discovered for doing this computation. However, it is known (Muroga [9]) that for linear functions

$$L_p < 2^{p^2}$$

which leads to the conclusion that this class is also realization sparse.

Thus this class can also be expected to exhibit error rates approaching 50 percent as p becomes large. The effect can be observed in the following table which was compiled from figures given for L_p by Muroga [9], page 821.

Number of inputs p	Number of realizable functions L	Total number of functions	Lower bound on worst case error D	Number of function values	Percent error
1	4	4	0	2	0
2	14	16	1	4	25
3	104	256	1	8	12
4	1882	65536	2	16	12
5	94572	4,294,967,296	5	32	16
6	15,028,134	1.844×10^{19}	12	64	19
7	8,378,070,864	3.403×10^{38}	29	128	23
8	17,561,539,552,946	1.158×10^{77}	70	256	27

Conjunctive Normal Form Expressions

Valiant [16] has studied the problem of learning k-conjunctive normal form Boolean expressions which have the form of a product of sums of k (or fewer) input variables. The model assumes examples of the behavior are presented according to a probability distribution and the major result is that if the target function is learnable, it will be identified with high probability using only a polynomial number on p of examples.

The class of functions realized by such expressions may be visualized as those functions which yield 1 on intersection of a number of regions on a Venn diagram of p dimensions where each region is the union of k (or fewer) input variables. If k=p then every possible function on p variables can be realized. Typically k will be less than p to reduce memory requirements and time needed to learn.

As with signature tables and threshold functions, the conjunctive normal form expressions form a realization sparse class. This can be seen by the following argument:

The number of conjuncts of length k on p input variables and their negations is $(2p)^k$. (Conjuncts of length less than k need not be considered because each one can be represented by a product of k length conjuncts. Thus, if k=2 one can represent x_1 as $(x_1+x_2)(x_1+x_2')$.) The number of different conjunctions is the number of subsets of $(2p)^k$ objects: $2^{(2p)^k}$. Not all of

these conjunctions yield unique functions so the number of functions is less than $2^{(2p)^t}$. $L_p < 2^{(2p)^t}$. Applying the definition given above, we see that this class is also realization sparse.

In Search of the Optimal Learning Machine: Truncation Machines

The above three sections describe the nature and cardinality of learnable classes for three kinds of learning machines. The question arises as to what learnable set of behaviors would be most desirable if one could specify them precisely. The measure to be considered here is the expected error rate assuming that the target behavior is selected from the set of all possible behaviors with each behavior being equiprobable and with all inputs to the target behavior being equiprobable. In other words, the following question is being asked: Find a set of L behaviors which will be the learnable set for the proposed machine and which will have the property that no other set of L vectors will have lower expected error (assuming uniform distribution on the target behaviors as mentioned above). Two solutions will be proposed to this problem, the *truncation machine* described here and the *G1 machine* described in the next section.

One obvious way to design the class of learnable vectors is to segment them into subvectors of some fixed length, say q, and require that all outputs within a segment be the same:

$$| \; xxxxxx \; | \; xxxxxx \; | \; \ldots\ldots \; | \; xxxxxx \; |$$

$$| \leftarrow q \rightarrow | \leftarrow q \rightarrow | \; \ldots\ldots \; | \leftarrow q \rightarrow |$$

As an example, suppose p=4 so that output vector length is 16. If q=4, the following sixteen vectors would compose the learnable set:

1. 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2. 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
3. 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
4. 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

5.	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
6.	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
7.	0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
8.	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
9.	1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
10.	1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1
11.	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
12.	1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1
13.	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
14.	1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
15.	1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
16.	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

It would appear that this set of L = 16 vectors uniformly span the space of all possible vectors of length 16 and that matching a target vector could be done optimally and in a straightforward manner. Suppose, as an example, that the target behavior is

0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1.

Then the learning machine could approximate this behavior with this vector:

0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0

And the rate of error would be 4 out of 16 or 25 percent. That is, each segment of q values is set to either 0 or 1 to match the majority of values in the target vector in that segment. If there are an equal number of 0's and 1's in the target vector for a given segment as occurs in the rightmost segment in the above example target, the learning machine can place either 0's or 1's in that segment with the same expected error rate.

The next question is whether one can build a machine with this class of learnable vectors. If $q = 2^r$ for some integer r, then such a machine is straightforward to build. Examining these learnable vectors, one can see that the output is independent of the rightmost r bits of the input and completely dependent on the remaining p-r inputs. So the desired learning machine is as shown in Figure 12, a system which ignores (or truncates) r inputs and keeps a table to specify

the output on the basis of the remaining inputs. Thus this is called the *truncation* machine.

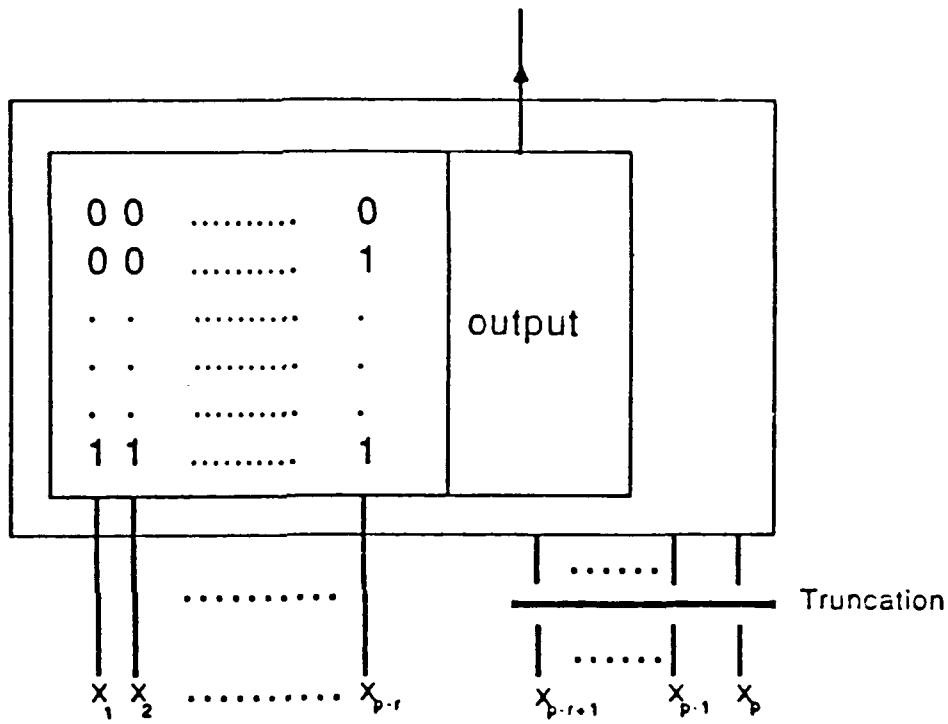


Figure 12. The truncation machine.

Next the characteristics of the truncation machine should be examined. The first observation is that the worst case error is n/2. By filling each segment with q/2 0's and q/2 1's, one can build the worst possible target vector for this machine. For the above example it would be:

0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

A worst case error of n/2 is a bad omen because most of the machines examined earlier had better worst case error characteristics.

What about the expected error for randomly selected target vectors? This can be computed by the formula

$$expected\ error = \frac{n}{2}\left[1 - \frac{1}{2^q}\binom{q}{\frac{q}{2}}\right]$$

which is derived in Appendix B. It turns out that this is not nearly optimal. In fact, in the case n=16, q=4, the average error is 5.0 whereas a set of L=16 vectors has been found with an average error of 4.27.

As will be seen below, the truncation machine is not even as good as the previously discussed machines from the point of view of minimum expected error. So an initial attempt to build a machine with low expected error has failed. The next section shows a more sophisticated approach.

In Search of the Optimal Learning Machine: The G1-Machine

If the uniform set of learnable vectors given above for the truncation machine is not distributed to achieve minimum expected error, what are the characteristics of such an optimum set? An example of a near optimal set was found in our studies and is given here, a set of L=16 vectors of length n=16 with average expected error of 4.27. That is, a randomly selected vector from n=16 space will be, on the average, a Hamming distance of 4.27 bits away from its nearest neighbor in the learnable set.

1.	0000 0000 0000 0000
2.	0001 0000 1111 1111
3.	0010 0111 0000 1111
4.	0011 0111 1111 0000
5.	0100 1011 0011 0011
6.	0101 1011 1100 1100
7.	0110 1100 0011 1100
8.	0111 1100 1100 0011
9.	1000 1101 0101 0101
10.	1001 1101 1010 1010
11.	1010 1010 0101 1010
12.	1011 1010 1010 0101
13.	1100 0110 0110 0110
14.	1101 0110 1001 1001
15.	1110 0001 0110 1001
16.	1111 0001 1001 0110

It is possible that this set is optimal and that no other set of 16 vectors exist with a lower expected error. We have not been able to prove this but we note that the achieved expected error is near the lower bound, 4.18, that can be computed from (5) above.

It would be desirable to be able to construct optimum sets for any n and L. Then the learning machine design procedure would follow these steps:

(1) Specify n and the desired maximum allowable expected error.

(2) Find the L required to reduce the expected error to required level.

(3) Synthesize L vectors which have minimum expected error and the machine to meet the specifications.

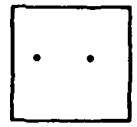
Unfortunately, there is no known method for finding such optimum sets except through completely enumerative methods. This section gives a method for constructing near optimal sets. The resulting machine will be called the G1 machine (named for the second author of this paper).

The construction method of the G1 machine will be first explained graphically and then in terms of binary vectors. The method assumes that $L = c\,2^h$ learnable vectors are to be spread

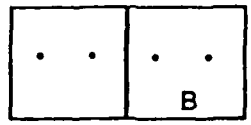
across the space of all behaviors where c and h are positive integers. The total space is broken into 2^h subspaces each with c learnable vectors. The construction method begins by placing c vectors optimally in one subspace as shown in Figure 13(a) (where $c = 2$). These optimal positions are found by enumerative or other means. Then the configuration is doubled as shown in (b). However, (b) can be improved by trying all possible rotations of the new space with respect to the old space to find the rotation which yields minimum expected error. Figure 13(c) shows such a rotation. The improvement comes from the fact that some vectors (those near B) are nearer a learnable vector after the rotation than before. The rightmost learnable vector in the original subspace is now nearer those points in B than any learnable vector in B's own subspace.

The construction continues as shown in Figure 13(d) and (e). The space is doubled again and rotated again for further reduction in expected error. This process continues until the complete behavior space has been accounted for. The complete synthesis procedure is near optimal because all the 2^h subspaces are locally optimal and they are rotated optimally with respect to each other. However, they are not globally optimal.

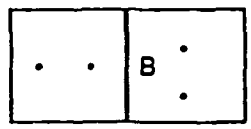
(a) A "basic space" with two optimally placed learnable vectors.



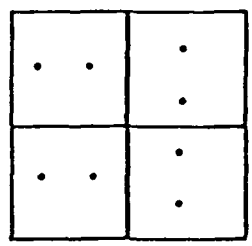
(b) Doubling the basic space.



(c) Rotating the new space to reduce expected error.



(d) Doubling again.



(e) Rotating again to further reduce expected error.

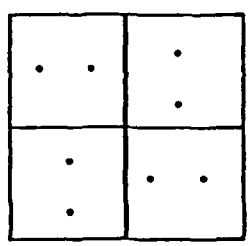


Figure 13. Constructing the G1 machine.

The G1 algorithm operates in the fashion illustrated in Figure 13 except that it creates a set of binary vectors which are designed to cover the behavior space nearly optimally. (The coding theorists [11] use a similar approach in their search for good codes.) Suppose, for example, that the initial space is the set of binary vectors of length three :000,001,...,111. Let us

choose a single vector that will cover this space optimally. Any vector will do, so let it be 000.

000

Then this set of learnable vectors is doubled by writing down two copies, one copy with a new zero added to the left end and one copy with a one added to the left end.

0000
1000

Next the new copy is *rotated* with respect to the old one as was done in Figure 13(c). That is, some of the columns in the new copy are inverted to reduce the average error of vectors of length four to their closest learnable vector in the set. There are eight possible inversions yielding the following possible learnable sets:

0000 1000	(no inversion)
0000 1001	(invert rightmost column in new space)
0000 1010	(invert second from rightmost column in new space)
- - -	
0000 1111	(invert all columns in new space)

Of the eight choices, an inversion is selected which achieves minimum expected error for the space of vectors of length 4. In this case, the vectors

0000
1011

were selected. This is a G1 machine for the space n=4.

For the space n=5, the next G1 machine is constructed by copying the above learnable set and again inverting columns in the second copy. The copy operation yields

00000
01011
10000
11011

and any of the three rightmost columns of the lower two vectors may be inverted. All such inversions are attempted and in each case, the average distance for a randomly selected vector of length five from one of these learnable vectors is computed. The rotation (or inversion) that yields the least average distance (or error) is selected. In this case, the rightmost and third from rightmost columns were inverted in the last two rows.

00000
01011
10101
11110

These four vectors are the learnable set for a G1 machine on the space n=5.

Repeating the construction for a third time yields a G1 machine for the case n=6. Copying yields eight vectors.

000000
001011
010101
011110
100000
101011
110101
111110

The best rotation of the rightmost three columns on the lower half of the vectors involves inverting the second and third bits from the right side.

000000
001011
010101
011110
100110
101101
110011
111000

Clearly the construction can be repeated to obtain G1 machines for any n-space. In this example, L is one eighth of the set of all vectors ($L = 2^n/8$) because the initial space had one learnable vector covering eight. In general, the ratio of L to 2^n can be any power of two depending on the selection of the initial space.

Figure 14 gives a plot of the expected error for many $G1$ machines in comparison with the known lower bound (5). The figure shows that these machines are either optimum or near optimum in all of the cases examined.

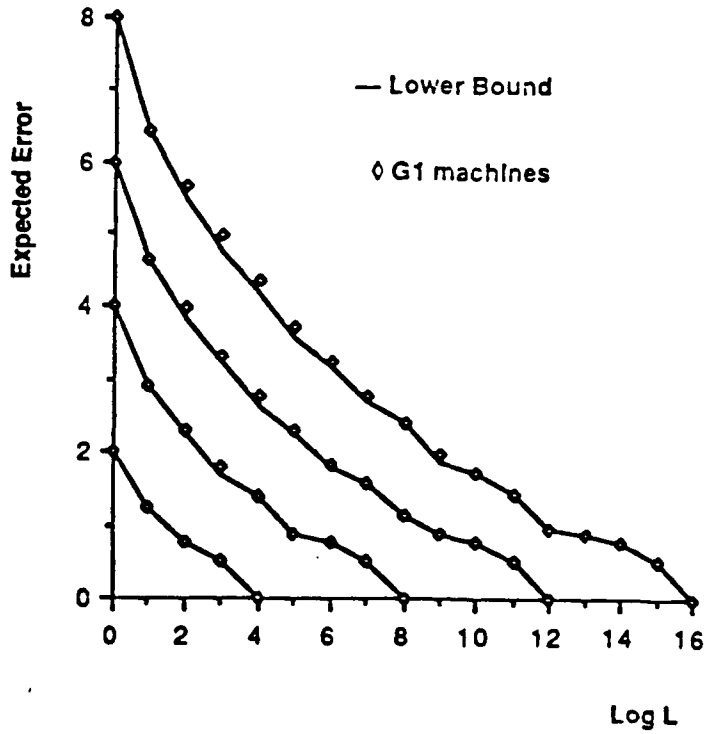


Figure 14. Expected error versus log L for the G1 machine as compared to the lower bound from (2). Curves are included for the cases n=4,8,12,16.

The G1 machine does not have practical significance since its only definition is that it is a set of learnable vectors. There is no known realization for such machines. However, theoreticians interested in minimizing expected error are bound to wonder what a set of optimally distributed vectors look like, and the G1 construction provides a way to generate near optimal sets if p is not too large.

The Random Vectors Machine

Another interesting way to construct a learning machine is to choose its learnable behaviors randomly. While this may not lead to practical devices, it provides a useful comparison for other machines. Specifically, what will be called the *random vectors machine* is a surprisingly good approximation to an optimum machine and one can easily compute its expected error. Thus the random vectors machine provides an easy way to compute an upper bound for the expected error of an optimum machine.

Assume then, that L vectors are selected independently and randomly from an equiprobable space of binary vectors of length n. These are to be the learnable vectors for the *random vectors machine* and learning will occur by having the machine choose which of its learnable vectors is closest to the target behavior.

Let $p_z(d)$ represent the probability that the nearest random vectors machine learnable vector to a randomly selected vector will be exactly d away. It can be shown that

$$p_Z(d) = \left[\frac{\sum_{i=0}^{n-d} \binom{n}{i}}{2^n} \right]^L - \left[\frac{\sum_{i=0}^{n-d-1} \binom{n}{i}}{2^n} \right]^L$$

The expected error for the random vectors machine is $\sum_{i=0}^{n} i \; p_Z(i)$ where p_Z is evaluated as given. As mentioned above, this formula gives an upper bound to and estimates well the

expected error for the optimum machine. Some of its values are plotted in the chart given in the following section.

Comparing Some Expected Errors

Figure 15 gives the expected error for the various machines discussed in this paper for the configuration $p=4$: signature table, threshold, k-CNF, truncation, G1, random vectors, and lower bound. As mentioned above, the G1 and the random vectors machines fairly well approximate the behavior of optimum machines. The G1 machines give a construction procedure for the learnable classes and the random vectors study provides an approximate calculation for the achievable minimum expected error. However, neither of these classes has a straightforward realization appropriate for applications.

The more traditional machines based on signature tables and threshold functions compare surprisingly well with optimum behavior often yielding error rates on the order of 20 to 30 percent above optimum in the ranges investigated. This indicates the learnable classes for these machines are fairly well distributed. Furthermore, no evidence came from this study to make one prefer one of these systems over any other on the grounds of expected error. The choice of which system to use should probably be dominated by the quality of the learning algorithms or the specific characterizations or properties of the learnable classes.

The worst behavior observed here is for the naive truncation machine which it was originally suggested might be optimal.

The lower bound curve has a scallop shape with one lobe for each value of D. The rightmost lobe corresponds to the case D=1 where every vector in the space is either learnable or one away from learnable. The second lobe from right graphs the case D=2 and so forth. The intersection points between the lobes are the places where (1) holds as an equality; the S_0 set around each v_0 has filled a sphere of size D and a larger S_0 set would move one into the case

D+1. These are the places where the coding theorists look for perfect codes.

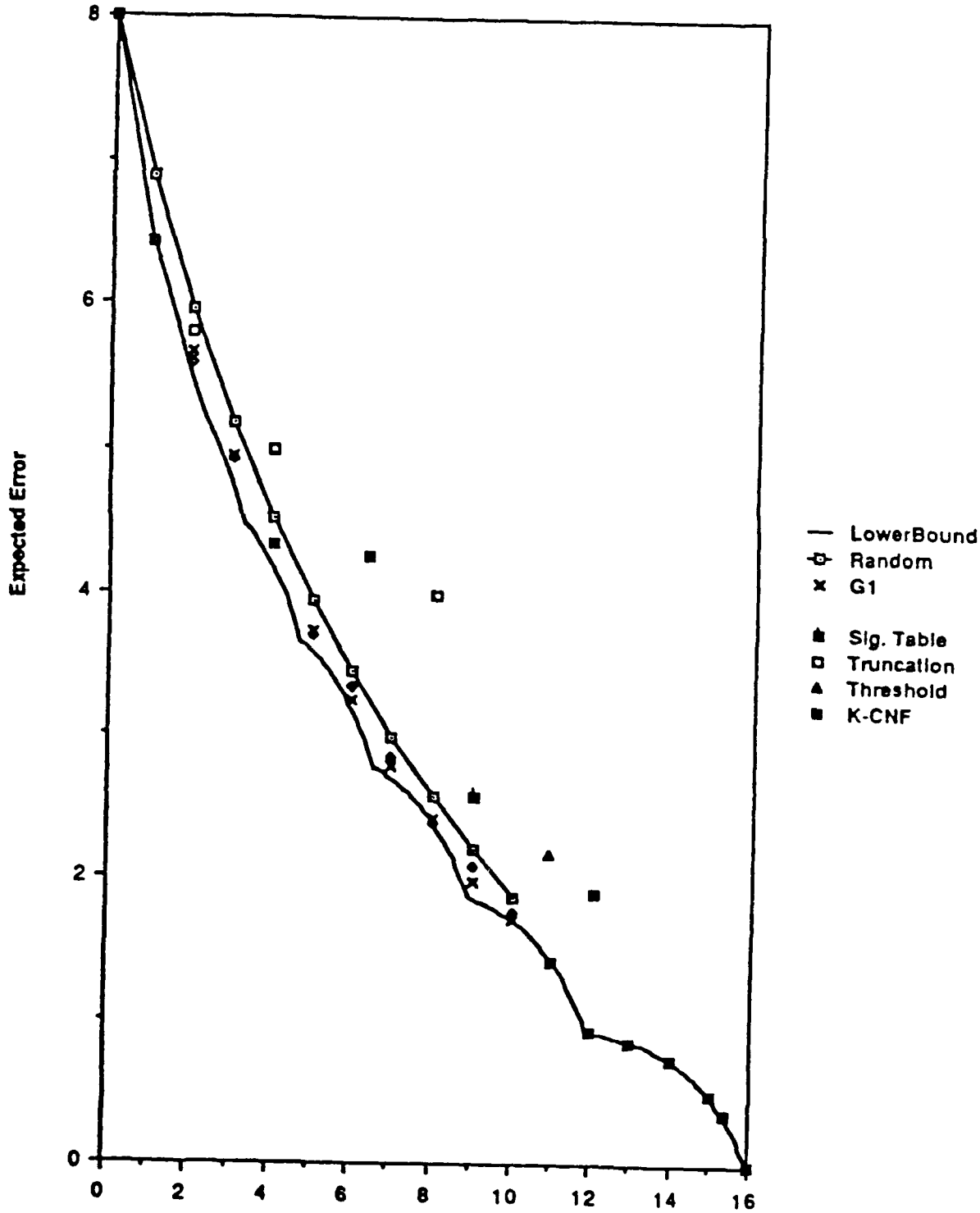


Figure 15. Expected error versus log L for the machines examined in this paper (n =16).

Conclusions

The main results of this paper are to emphasize the fundamental difficulty of learning. If the learning is to be done on the basis of a reasonable amount of information, then the learning machine will be realization sparse and learned behaviors will typically be only small improvements over a random decision maker. If the target behavior is to be learned accurately, the learning requires the observation of a very large fraction of the set of all possible behaviors. The machine acts more or less as if it were doing rote memorization. Relationship (4) gives a lower bound on the number $\log_2(L_p)$ of needed observations required to achieve accuracy λ

Many well known learning machines are realization sparse as is the case for the signature tables, linear models, and conjunctive normal form machines examined here. These machines are popular because of their fast and effective learning algorithms but the number of behaviors that they can learn is relatively small. Thus, it was possible for Minsky and Papert [8] to prove a large number of noncomputability results for the perceptron. Presumably similar results could be proven for any of the other realization sparse classes. The only way such machines can function very effectively is if the class of target behaviors corresponds closely to the class of learnable behaviors.

While the realization sparse machines can learn few functions precisely, they can converge toward many functions at least to some degree. Thus Samuel [13,14], Sejnowski [15], and others have observed improvements in system behaviors using these models. However, the results here make one suspect that there probably were vastly more accurate behaviors in their respective function spaces, but they did not have the means to converge upon them.

Human learning probably does not include the capability to acquire general functions unless p is effectively very small, say 2 or 3. Human learning probably depends heavily on rote memorization and the ability to recognize perturbations on the original patterns.

An interesting dimension for studying classes of learnable behaviors relates to their distribution across the total space of behaviors. It was determined here that for small p, the specific models studied (signature tables, linear models, conjunctive normal form systems) deviate substantially from uniform distributions. This opens the question as to whether machines with lower expected errors might be found, and the G1 machine exhibits the type of learnable behaviors that would achieve this goal.

The implications of this work for the many studies now ongoing in machine learning (see, for example [2,6,7]) are a matter for additional research. The emphasis here is that there is a necessary tradeoff between rate of learning and achievable accuracy for any learning machine. In some cases, it is possible to measure these parameters accurately and to understand where a particular machine lies on the spectrum. Until these issues are more fully understood, the design of learning systems will remain a "black art".

Appendix A

Counting Signature Tables

This paper and earlier papers [1,14] have shown the type of functions that can be represented (or learned) by signature tables. It is clear that many functions cannot be represented if the sizes of internal alphabets are restricted and the purpose of this section is to evaluate how serious the limitations are. The methodology given here is an improved and simplified version of that given in Biermann *et al.* [1]. If one can compute the size L of the learnable class, it becomes possible to estimate the probability that a randomly selected behavior can be learned. One can also use L in formula (1) to obtain a lower bound on the worst case error that could possibly be encountered.

One can compute the number L of functions computable by the table system of Figure 11 assuming $v_i \leq 2$ for $i = 1,2,\ldots,6$, and in the process derive a general methodology for handling any such table system. It turns out that one cannot simply compute the number of settings of all of the 28 binary constants in the table, a total of 2^{28}, because a single function may be realizable by many different settings. So one must count only nonredundant settings.

Consider, as a start, the portion of the tree labelled f_3. The output alphabet size v_3 could equal one in which case the output column would be either (0,0,0,0) or (1,1,1,1). But the class of functions computable by the whole signature table system is the same regardless of which is selected. That is, the output leaving f_3 is simply an internal code for the system and 0 and 1 are abstract symbols. Adjustments can be made to the top table of f_1 to obtain the same system behavior regardless of whether the f_3 output vector is (0,0,0,0) or (1,1,1,1). Thus one can conclude that the number of nonredundant configurations of f_3 with output alphabet of size one is 1 and this is written as

$$N(f_3,1) = 1.$$

If v_3=2 then fourteen vectors are possible, (0.,0,0,1), (0,0,1,0),...,(1,1,1,0). Again, as explained in [1], since 0 and 1 are abstract symbols with only internal meaning, the vectors (0,0,0,1) and (1,1,1,0) result in identical classes of behaviors so only one of them is counted. In fact, only half of the fourteen vectors are nonredundant: (0,0,0,1) (0,0,1,0) (0,0,1,1) (0,1,0,0) (0,1,0,1) (0,1,1,0) (0,1,1,1). Thus $N(f_3,2)$=7. In general, the number $N(t,v)$ of configurations of t with q binary inputs and output vocabulary of size v when t is a single bottom level table is

$$N(t,v) = \begin{cases} 0 & \text{if } 2^q < v \\ \dfrac{\sum_{i=0}^{v} (-1)^i \binom{v}{i} (v-i)^{2^q}}{v!} & \textit{otherwise} \end{cases} \tag{3}$$

as derived in [1].

Consider next the subtree f_1 in Figure 11. First the number $C(f_1,v,z)$ of nonredundant configurations of the top table in f_1 will be computed as a function of v, the output vocabulary size and z, a vector giving the sizes of all the input vectors to the top table of f_1. Then the number $N(t,v)$ of nonredundant configurations for the whole f_1 tree will be determined. Let b(t) stand for the branching factor below the top table in tree t. Then z will be a vector of length b(t). $b(f_1)$=2 in Figure 11.

Assuming that the highest table in f_1 is not the top table or a bottom table in the complete table system and that this highest table has input and output vocabularies of size two, it is shown in [1] that

$$C(f_1,2,z) = \begin{cases} 0 & \text{if } q=0 \\ \frac{1}{2}\sum_{i=0}^{q} (-1)^i \binom{q}{i} 2^{2^{(q-i)}} & \textit{otherwise} \end{cases}$$

where q is the number of binary inputs. In the case, q = 2 and

$$C(f_1,2,(2,2)) = 5.$$

The five nonredundant vectors are (0,0,0,1), (0,0,1,0), (0,1,0,0), (0,1,1,0), (0,1,1,1). (The vector (0,1,0,1) is redundant because it yields the same value regardless of the output of f_3. (0,0,1,1) is redundant because it yields the same value regardless of the value of f_4.)

The number of nonredundant subtrees f_1 with output alphabet of size v and input alphabets from subtrees f_3 and f_4 of sizes i_3 and i_4 is the product of the number of nonredundant top tables times the product of the numbers of nonredundant subtrees.

$$C(f_1,v,(i_3,i_4))N(f_3,i_3)N(f_4,i_4)$$

But the inputs to the top table may have alphabet sizes of 1 or 2, so $N(f_1,v)$ is the sum over all such values.

$$N(f_1,v) = \sum_{\substack{all\,(i_1,i_2)\\ 1\le i_1\le 2\\ 1\le i_2\le 2}} C(f_1,v,(i_1,i_2))N(f_3,i_3)N(f_4,i_4))$$

More generally a tree t may have a branching factor of b(t) below the top table and input alphabets to the top table from 1 to v_m. The counting formula is

$$N(t,v) = \sum_{\substack{all\\ b(t)\text{-tuples of}\\ \text{of entries } j\\ 1\le j\le v_m}} C(t,v,z)\prod_{h=1}^{b(t)} N(subtree(h,t),entry(h,z)) \tag{4}$$

where subtree (h,t) returns the h-th child below the highest table in t and entry (h,z) selects the h-th entry in z.

Equation (4) counts the number of distinct functions computable by any subtree t in a signature table system except for (a) the case where t is a bottom subtree which includes only one table (and (3) is used) and (b) the case where t is the whole signature table system (and a

special form for C(t,v,z) is used). Applying (4) to f_1 yields

$$
\begin{aligned}
N(f_1,2) = {} & C(t,2,(2,2))N(f_3,2)N(f_4,2) \\
& + C(t,2,(1,2))N(f_3,1)N(f_4,2) \\
& + C(t,2,(2,1))N(f_3,2)N(f_4,1) \\
& + C(t,2,(1,1))N(f_3,1)N(f_4,1) \\
& = 5\cdot7\cdot7+1\cdot1\cdot7+1\cdot7\cdot1+0\cdot1\cdot1 = 259
\end{aligned}
$$

In the case of output vocabulary of size one, (4) yields

$$
\begin{aligned}
N(f_1,1) = {} & C(t,1,(2,2))N(f_3,2)N(f_4,2) \\
& + C(t,1,(1,2))N(f_3,1)N(f_4,2) \\
& + C(t,1,(2,1))N(f_3,2)N(f_4,1) \\
& + C(t,1,(1,1))N(f_3,1)N(f_4,1) \\
& = 0\cdot7\cdot7+0\cdot1\cdot7+0\cdot7\cdot1+1\cdot1\cdot1 = 1
\end{aligned}
$$

The top table in Figure 11 may have, according to [1],

$$C(f,v,z) = \sum_{i=0}^{q} (-1)^i \binom{q}{i} v^{2(q-i)}$$

where q is the number of binary inputs. In this example, v=2 and q=2, yielding

$$C(f,2,(2,2)) = 10.$$

So the total number of functions representable by f with an output vocabulary of size 2 is

$$
\begin{aligned}
N(f,2) = {} & C(f,2,(2,2))N(f_1,2)N(f_2,2) \\
& + C(f,2,(2,1))N(f_1,2)N(f_2,1) \\
& + C(f,2,(1,2))N(f_1,1)N(f_2,2) \\
& + C(f,2,(1,1)N(f_1,1)N(f_2,1) \\
& = 10\cdot259\cdot259+2\cdot1\cdot259+2\cdot259\cdot1+2\cdot1\cdot1
\end{aligned}
$$

$= 671{,}848$

For output vocabulary size of one,

$$N(f\ ,1) = 2$$

This means that the total number of functions representable by the system of Figure 11 under the specified vocabulary limitations is $L = N(f\ ,1) + N(f\ ,2) = 671{,}850$. Comparing L with the total number of functions on eight variables, $2^{2^8} = 2^{256} = 1.16 \times 10^{77}$, one can see that only a very small fraction of the set of all functions can be learned. Furthermore, using (1) it is possible to find a lower bound on the worst error D that could occur in trying to learn a target behavior. It yields $D \geq 91$ which means that there exist target functions in which the best learning algorithm may produce errors on 91 out of 256 possible inputs. Considering that a random decision maker would be wrong only 128 out of 256 times, one can see that the system is capable of very poor performance.

Severe as these effects may be, they are exponentially worse in deeper trees. This can be seen by bounding the counting functions given above and deriving a general bound for number of functions versus the depth of the tree. Define the depth d to be the number of levels of tables in binary table systems of the form of Figure 11. (Thus d=3, for Figure 11.) Consider such tables of any depth $d > 1$ with all vocabulary sizes limited to two or less.

For bottom level tables f_{bottom}, as above, $N(f_{bottom}, 2) = 7$. For intermediate level tables f_{level},

$$N(f_{level}, 2) = 5[N(f_{level-1}, 2)]^2 + small\ \ terms$$

Similarly, for the top level function

$$N(f_{top}, 2) < 11[N(f_{top-1}, 2)]^2.$$

Applying these at several levels yields

$d=2 \quad L < 11(7^2)$

$d=3 \quad L < 11(6(7)^2)^2$

$d=4 \quad L < 11(6(6(7)^2)^2)^2$

$d=5 \quad L < 11(6(6(6(7)^2)^2)^2)^2$.

In general,

$$L < 11(6^{\sum_{i=1}^{d-2} 2^i} 7^{2^{d-1}})$$

One can replace $\sum_{i=1}^{d-2} 2^i$ with $2^{-2}2^{d-1}$ to obtain

$$L < 11(6^{-2})(6^{2^{d-1}} 7^{2^{d-1}}) = \frac{11}{36}(42^{2^{d-1}}).$$

thus

$$L < \frac{1}{3}(42^{2^{d-1}}) < \frac{1}{3}[2^{6(2^{d-1})}]$$

But the total number of functions for such a table system of depth d is $2^{2^{2^d}}$. The ratio of realizable functions to actual functions is thus

$$\frac{1}{3}\frac{[2^{6(2^{d-1})}]}{2^{2^{2^d}}}$$

This reduces to

$$\frac{1}{3}[2^{3(2^d)-2^{2^d}}]$$

If d is greater than 2, the ratio is less than

$$\frac{1}{3(2^{2^{(2^d-1)}})}.$$

So the ratio of the number of realizable functions to actual functions decreases dramatically as d increases.

Appendix B

The Average Error for Truncation Machines.

The average error for truncation machines can be computed by the formula

$$average\ error = \frac{\sum_{all\ vectors} (size\ of\ error)}{number\ of\ vectors}$$

Consider first the problem of computing average error for a single segment of q identical bits. Then the total average error will be obtained by multiplying by the number of segments, n/q.

$$average\ error\ per\ segment = \frac{\sum_{i=0}^{q} (number\ of\ segments\ with\ i\ 1's) \times (error\ if\ i\ 1's)}{2^q}$$

$$average\ error\ per\ segment = \frac{1}{2^q}\Big[\sum_{i=0}^{\frac{q}{2}-1} (number\ of\ segments\ with\ i\ 1's) \times i$$

$$+ (number\ of\ segments\ with\ \frac{q}{2}\ 1's) \times \frac{q}{2}$$

$$+ \sum_{i=\frac{q}{2}+1}^{q} (number\ of\ segments\ with\ i\ 1's) \times (q-i)\Big]$$

$$= \frac{1}{2^q}\left[\sum_{i=0}^{\frac{q}{2}-1} \binom{q}{i} i + \binom{q}{\frac{q}{2}} \frac{q}{2} + \sum_{i=\frac{q}{2}+1}^{q} \binom{q}{i}(q-i)\right]$$

$$= \frac{1}{2^q}\left[\sum_{i=0}^{\frac{q}{2}-1} \binom{q}{i} i + \binom{q}{\frac{q}{2}} \frac{q}{2} + \sum_{i=0}^{\frac{q}{2}-1} \binom{q}{i} i\right]$$

$$= \frac{1}{2^{q-1}}\left[\sum_{i=0}^{\frac{q}{2}-1} \binom{q}{i} i + \binom{q}{\frac{q}{2}} \frac{q}{4}\right]$$

$$= \frac{1}{2^{q-1}} \left[\sum_{i=0}^{\frac{q}{2}-1} \binom{q}{i} i + \binom{q}{\frac{q}{2}} \frac{q}{2} - \binom{q}{\frac{q}{2}} \frac{q}{4} \right]$$

$$= \frac{1}{2^{q-1}} \left[\sum_{i=0}^{\frac{q}{2}} \binom{q}{i} i - \binom{q}{\frac{q}{2}} \frac{q}{4} \right]$$

(But for q even, the identity

$$\sum_{i=0}^{\frac{q}{2}} \binom{q}{i} i = q\, 2^{q-2} \; can \;\; be \;\; used.)$$

$$= \frac{1}{2^{q-1}} \left[q\, 2^{q-2} - \frac{q}{4} \binom{q}{\frac{q}{2}} \right]$$

$$= \frac{q}{2} \left[1 - \frac{1}{2^q} \binom{q}{\frac{q}{2}} \right]$$

This is multiplied by the number of segments n/q to obtain

$$average \;\; error = \frac{n}{2} \left[1 - \frac{1}{2^q} \binom{q}{\frac{q}{2}} \right]$$

References

1. Biermann, A.W., Fairfield, J. and Beres, T., Signature Table systems and learning, *IEEE Transactions on Systems, Man and Cybernetics,* vol. 12, SMC-12, No. 5, Sept./Oct. 1982.

2. Biermann, A.W., Guiho, G., and Kodratoff, Y., (Eds.), *Automatic Program Construction Techniques,* Macmillan, 1984.

3. Curtis, H.A., *A New Approach to the Design of Switching Circuits,* D. Van Nostrand, Princeton, NJ, 1962.

4. Hamming, R.W., *Coding and Information Theory,* Prentice Hall, 1980.

5. McClelland, J.L., Rumelhart, D.E., and the PDP Research Group, *Parallel Distributed Processing,* Volume 2, MIT Press, Cambridge, Mass. 1986.

6. Michalski, R.S., Carbonell, J.G., Mitchell, T.M., (Eds.), *Machine Learning,* Springer-Verlag, 1984.

7. Michalski, R.S., Carbonell, J.G., Mitchell, T.M., (Eds.), *Machine Learning, Vol. II,* Morgan Kaufmann, 1986.

8. Minsky, M. and Papert, S., *Perceptrons,* MIT Press, Cambridge, MA, 1969.

9. Muroga, S. *Threshold Logic and its Applications,* John Wiley and Sons, Inc., 1971.

10. Nilsson, N.J., *Learning Machines,* McGraw-Hill, New York, 1965.

11. Peterson, W.W. *Error Correcting Codes,* MIT Press, 1961.

12. Rumelhart, D.E., McClelland, J.L. and the PDP Research Group, *Parallel Distributed Processing,* Volume 1, MIT Press, Cambridge, Mass. 1986.

13. Samuel A., Some studies in machine learning using the game of checkers, *IBM J. Res. Develop.,* vol. 3, 1959, pp. 221-229, reprinted in Feigenbaum E. and Feldman, J., Eds., *Computers and Thought,* McGraw-Hill, New York 1963, pp. 71-105.

14. Samuel A., Some studies in machine learning using the game of checkers II. Recent progress, *IBM J. Res. Develop* , vol. 11, 1967, pp. 601-617.

15. Sejnowski, T., Language learning in Massively-Parallel Networks, *Proceedings of the 24th Annual Meeting of the Association for* Computational Linguistics, 1986, pp. 184.

16. Valiant, L., A Theory of the Learnable, *Communications of the ACM,* vol. 27, No. 11, 1984.