A basic approach to pattern recognition

By J. R. Ullmann*

In looking for a plausible alternative to the representation of pattern similarity by a metric, there are two main problems. The first is to find a ranking of patterns in order of a similarity, and the second is to store this ranking economically. This paper deals only with the second of these problems, by means of a successive approximation technique which has been tested by computer simulation.

1. Introduction

A binary pattern is an array of 0s and 1s, and changing the state of any digit makes a different pattern. We say that variants of a character are patterns for which a character-recognition machine is required to give the same output; assuming, for example, that different output signals are required for a given character in upper and lower case.

A character-recognition machine designer does not generally have available a complete list of all the variants of every character which his machine is to recognize. He may therefore attempt to design a machine which, having been shown a number of correctly labelled examples of variants (which we call *paradigms*) of characters, will predict the correct output signal for a new input pattern.

The basic idea which has been used for such predictions is to treat an input pattern as a variant of the character to whose paradigms it is most similar. Patterns are conveniently represented by points in a space chosen so that effective similarity is a function of distance: the choice of suitable metrics is discussed by Sebestyen (1962). The hypothesis that suitable spaces and metrics can in fact be found (cf. "the compactness hypothesis", (Aizerman, 1963)) has been implemented in a number of ways.

For example, certain information may be extracted from a pattern and expressed as a smaller pattern or code word which is compared with paradigm code words according to some measure of similarity, derived from the chosen metric of the code word space. The transformation applied to patterns to reduce them to code words may be fixed (Grimsdale, Sumner, Tunis and Kilburn, 1959; Clowes and Parks, 1961) or may be optimized automatically (Uhr and Vossler, 1961; Kamentsky and Liu, 1963). It is, of course, also necessary to choose paradigms, and these may be stored explicitly, or in the form of some sort of average for each character. Barus (1962) proposes explicit storage so that for a given character, paradigm variants more than a certain amount dissimilar to most of the other paradigm variants may be rejected.

The standpoint of the present paper is not that the idea of representing similarity by a metric has been weighed in the balance and found wanting, but that it is not clear whether this idea is the most practical basis for character recognition in the case where very many characters are to be recognized, and each has very many variants. Whatever the difficulty, an alternative basis should therefore be sought, in the course of open-minded scientific procedure.

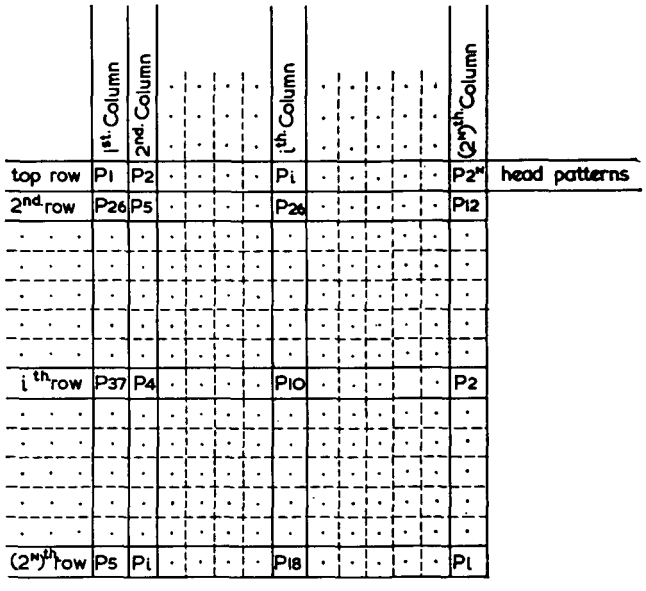

Fig. 1.—Ranking of patterns in order of similarity

In Fig. 1 the top row comprises all possible N-bit binary patterns, $P_1, P_2, \ldots, P_i, \ldots, P_{2^N}$. The columns of Fig. 1 each contain 2^N different N-bit patterns ranked in order of similarity to the pattern at the head of the column (the *head-pattern*). In any column in Fig. 1 the pattern in the ith row is more similar to the head-pattern than is the pattern in the $(i + 1)$th row, and so on. (It is generally not true that in any column the pattern in the $(i + 1)$th row is more similar to the pattern in the ith row than is the pattern in the $(i + 2)$th row.) The assumption that no two patterns are equally similar to a head-pattern is a convenient simplification. The advantage of the explicit ranking of Fig. 1 over metrical representation of similarity is of course generality, in that Fig. 1 can represent rankings which can not be represented by metrics.

**National Physical Laboratory, Teddington, Middx.*

This paper is concerned with a facile example of a character-recognition scheme employing explicit ranking. In this example, which we call *scheme A*, a number of labelled paradigm variants are stored. For any new input pattern, scheme A, taking the input pattern as the head-pattern, looks up in Fig. 1 the most similar stored paradigm, and gives the output signal associated with this paradigm. (To say instead, for example, that scheme A should give the output signal corresponding to the character with most paradigms with greatest similarity to the present input pattern would be an unwarranted complication at this stage.) As many paradigms are stored in scheme A as are found necessary for correct recognition of patterns.

Fortunately, for the purposes of this paper it is not necessary to give a rigorous definition of similarity. But to be specific we say that similarity ranking is a Fig. 1 ranking which gives the best practical performance of scheme A. The essential point of our approach is that we are not bound to rankings for which a representation metric can be found.

Scheme A presents two main problems. The foremost of these is how to obtain the Fig. 1 information, and the second is how to store this information economically, having obtained it: this paper deals only with the second of these problems.

If instead of having 2^N rows in Fig. 1, all but the top k rows are discarded, the result is that for input patterns more than a certain amount dissimilar to any paradigm, scheme A will find no output signal. This seems realistic, and so we are left only with the non-trivial problem of effectively reducing the number of columns of Fig. 1, without knowing any paradigms of the characters to be recognized. (If the paradigms were known, and the lower rows of Fig. 1 had been discarded, then of course some of the columns of Fig. 1 could simply be discarded.) A successive approximation technique for Fig. 1 column economy is employed in a scheme which we call *scheme B*, which is introduced in the following pages. Apart from the use of this economy technique, scheme B is the same as scheme A. A comparison of results of computer simulations of schemes A and B with arbitrary (but not purely random) rankings provides a quantitative indication of the amount of error which the economy technique introduces. As far as possible pseudo-random patterns were used in the simulation, and this, though far removed from the practical problem of character recognition, has the advantage of reproducibility.

pattern	1	1	0	1	0	0	1	0	1	0
A tick in this row indicates bit belongs to example n-tuple		✓			✓			✓	✓	
n-tuple state		1			0			0	1	

Fig. 2(a).—Example of an *n*-tuple state

2. Economy by consistency technique

(*a*) *n-tuples*

We say that a pattern digit is the *state* of a *bit*. Thus in the three digit pattern 101, the three bits are in states 1, 0, 1, respectively. An *n-tuple* is a set of n bits, and the state of these n bits is the *n-tuple state*. Fig. 2(a) shows an example of this for a ten-bit pattern.

Scheme B employs n-tuples which are chosen, for simplicity, so that

(i) the value of n is the same for all n-tuples;
(ii) the number of n-tuples to which a bit belongs is the same for every bit. In fact we choose to set this number equal to n, so that altogether N n-tuples are chosen.

Apart from these arbitrary constraints the choice of n-tuples is random. In the following pages states of these chosen n-tuples will be referred to simply as *n-tuple states*, and states of other n-tuples will not be mentioned.

(*b*) *Ranking in scheme B*

Let us consider only the top row (the head-patterns) and, say, the ith row of Fig. 1. We say that patterns in these two rows which are also in the same column of Fig. 1 are *corresponding patterns*. States of the same n-tuple in corresponding patterns are *corresponding n-tuple states*. Where a given n-tuple state occurs in different head-patterns, the corresponding n-tuple state in the ith row of Fig. 1 is generally different. Fig. 2(b) illustrates this for the head-pattern 3-tuple state –1–00, where "–" stands for digits not belonging to the n-tuple, and $N = 5$.

We say that the state of bits not belonging to an n-tuple is the *context* of that n-tuple. For each head-pattern n-tuple state, scheme B stores all the correspond-

top row	00000	10000	01000	11000	00100	10100	01100	11100	. . . .
i^{th} row	01100	01000	10010	11100	01001	11100	00010	01101	. . . .
i^{th} row n-tuple state corresponding to head pattern n-tuple state -1-00			-0-10	-1-00			-0-10	-1-01	. . . .

Fig. 2(b).—Examples of corresponding *n*-tuple states

ing ith row n-tuple states, irrespective of context. For example, for the example of Fig. 2(b) the stored information specifies that the ith row n-tuple states –0–10, –1–00, –0–10, –1–01 correspond to the head-pattern n-tuple state –1–00, but information specifying the context in which these n-tuple states occurred is not stored.

In scheme B the ith row of Fig. 1 is replaced by $N \times 2^n$ lists of ith row n-tuple states which correspond to the $N \times 2^n$ head-pattern n-tuple states. Let h be the average number of entries in each of these lists, i.e. the average number of ith row n-tuple states corresponding to a head-pattern n-tuple state. Then the number of n-tuple states stored in scheme B to replace the ith row patterns of scheme A is

$$N \times 2^n \times h$$

while the number of patterns in the ith row of Fig. 1 is 2^N. To obtain a rough indication of the economy achieved in scheme B we compare these two numbers and see that there is economy when

$$h < 2^{(N-n)}/N.$$

(The ith row information stored in scheme B is that which would be stored in a consistency pattern association network (Ullmann, 1962) after all the corresponding head-patterns and ith row patterns had been presented to it in turn.)

(*c*) *Paradigm storage in scheme B*

Having said how the Fig. 1 information is stored, we describe the storage of paradigms in scheme B, but it is helpful first to consider paradigm storage in scheme A. When a paradigm is presented for storage, scheme A stores this paradigm pattern and associated output, and also makes a mark in Fig. 1 wherever this particular paradigm pattern occurs. The stored paradigm most similar to a presented input pattern is in Fig. 1 the marked pattern nearest the head-pattern in the column for which the head-pattern is the input pattern. In scheme B when a paradigm is presented for storage, if all its n-tuple states belong to patterns in the ith row of Fig. 1, then we mark all these n-tuple states by entering them in the set R_i which is defined in the following paragraph. This storage procedure is used for all values of i. The output associated with each paradigm is also stored.

In scheme B we say that ith row n-tuple states which correspond to at least one head-pattern n-tuple state belong to the set Q_i. If all the n-tuple states included in a presented paradigm belong to Q_i, then we say that these n-tuple states belong to the set R_i. If this condition is fulfilled, and a record of the members of R_i is stored in the scheme, we say that the paradigm is *stored in the ith row*. Thus if any n-tuple states in a presented paradigm do not belong to Q_i, this paradigm is not stored in the ith row. R_i is the union of the sets of n-tuple states in stored paradigms: thus if one paradigm has already been stored in the ith row, and another paradigm is presented, such that all its n-tuple states belong to Q_i, then these states are entered in the set R_i.

If, after presentation, a paradigm is not stored in the ith row in scheme B, this indicates that this paradigm does not occur in the ith row of Fig. 1 in any column. If a paradigm is stored in scheme B but not included in the ith row of Fig. 1, this causes errors which are reflected in the quantitative results of computer simulation of scheme B.

Suppose that h, the average number of ith row n-tuple states corresponding to any head-pattern n-tuple state has the value 2^n. In this case, many patterns which do not occur in the ith row of Fig. 1 can be stored as paradigms in the ith row of scheme B. This difficulty imposes a restriction on the maximum tolerable value of h, and for this reason a number of different values of h were tried out in computer simulation.

(*d*) *Operation of scheme B*

When a number of paradigms have been stored in the ith row and a new input pattern is presented to scheme B, scheme B finds the stored paradigm most similar to the present input pattern as follows. It is again convenient to consider the ith row. We define S_{i0} as the set of n-tuple states which correspond to the n-tuple states in the presented pattern and which also belong to R_i. In scheme B an N-digit pattern F_{i0} is now constructed according to the following rule. If a given bit, d, is in state 1 in at least one n-tuple state, belonging to S_{i0}, of each of the n chosen n-tuples to which d belongs, then d is in state 1 in F_{i0}. Similarly, if d is in state 0 in at least one n-tuple state, belonging to S_{i0}, of each of the n chosen n-tuples to which d belongs, then d is in state 0 in F_{i0}. Thus in F_{i0} d could be in state 1, state 0, state "neither 0 nor 1," or state "both 1 and 0." This rule for determining the state of d in F_{i0} applies similarly for the states of all the other digits in F_{i0}.

We define S_{i1} as the set of n-tuple states which belong to S_{i0} and which occur in F_{i0}. Scheme B now constructs a pattern F_{i1} from S_{i1} according to the same rule that was used in constructing F_{i0} from S_{i0}. S_{i2} is the set of n-tuple states which belong to S_{i1} and which occur in F_{i1}. S_{i3}, S_{i4}, etc., and F_{i2}, F_{i3}, F_{i4}, etc., are similarly defined. Scheme B constructs in turn F_{i0}, F_{i1}, F_{i2}, etc., and eventually a terminal pattern F_{iw} which is defined as the first pattern in the series F_{i0}, F_{i1}, . . . such that

$$F_{iw} = F_{i(w+1)}.$$

(This iterative process is bound to terminate because in $S_{i0}, S_{i1}, S_{i2}, \ldots$, any set S_{ij} includes the set $S_{i(j-1)}$, and the number of members of S_{i0} is finite.)

For $N = 5$, $n = 2$, for the input pattern 10011 the set S_{i0} might, for example, be that shown in Fig. 3(*a*). In the pattern F_{i0}, which has been computed from S_{i0} according to the above rule, bits in state "both 1 and 0" are represented by $\binom{1}{0}$. Figs. 3(*b*), (*c*), (*d*) show S_{i1} and F_{i1}, S_{i2} and F_{i2}, S_{i3} and F_{i3}, respectively.

Scheme B finds the terminal patterns F_{iw} for all i from 1 to k, and takes as the paradigm most similar to the

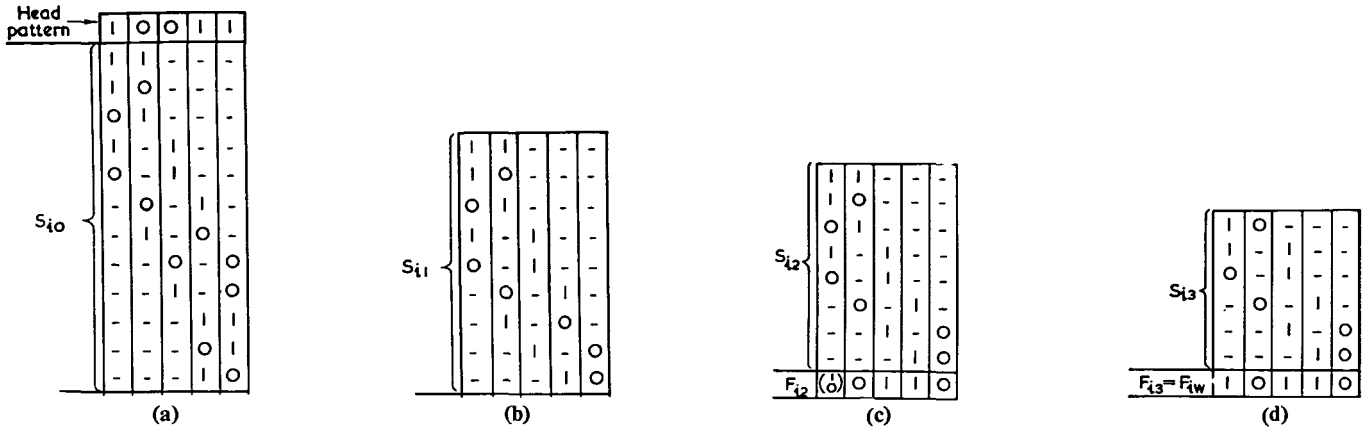

Fig. 3.—Example of computation of F_{iw}

present input pattern the pattern F_{iw} for the least value of i such that F_{iw} does not consist entirely of bits in state "neither 0 nor 1". The results of computer simulation scheme B show that the pattern thus selected is in fact, at least in a number of cases, the paradigm most similar to the presented pattern.

(One may say that a terminal pattern is arrived at by successive approximation because in the successive patterns $F_{i0}, F_{i1}, \ldots, F_{iw}$, the number of bits in state "both 0 and 1" generally decreases. If a bit is in this state in F_{ij} it may either be in state 0 or state 1 in $F_{i(j+1)}$ and it is this lack of specificity which leads us to think loosely of F_{ij} as an approximation for $F_{i(j+1)}$.)

(*e*) *Computer simulation of scheme B*

The simulation compared the performance of scheme A with that of scheme B when dealing with the same patterns. The values $N = 10$, $n = 4$ were chosen, being the maximum practicable on the ACE computer.

The first step was to choose the n-tuples, as specified in section (*b*) above. The pseudo-random number generator used in this and throughout the simulation was

$$x_{K+1} = (2^{13} - 3)x_K - 1, \text{ in mod } 31.$$

The next step was to choose for each head-pattern in turn a corresponding ith row pattern. The algorithm which made these choices ensured that each choice introduced on average only a small and (loosely) controlled number of new members to the set Q_i. This algorithm thus determined the final value of h. Actually only one row, say the ith, was simulated, since nothing would have been gained by simulating more rows.

Pseudo-random patterns were then generated and examined until one was found such that all the 10 n-tuple states in the pattern belonged to Q_i. The successful pattern was stored as a paradigm—i.e. its n-tuple states were listed as members of R_i—and also the pattern was entered in a list of stored paradigms.

Next scheme B was presented in turn with sixty-four randomly chosen input patterns (head-patterns), and for these ith row patterns (i.e. F_{iw} patterns) were found by iterative operation as described above. If for a given head-pattern the corresponding scheme A and scheme B ith row patterns were not identical, even if only one bit in the scheme B pattern was in state "both 0 and 1", this was counted as an *error*. If the ith row patterns were identical, their membership of the list of stored paradigms was checked (in the experiment they were always found present in this list) and this event was counted as a *correct paradigm output*. For the sixty-four trial head-patterns the total number of errors and correct paradigm outputs were found. Altogether four paradigms were stored, the scheme being tested with sixty-four trial head-patterns after the storage of each paradigm. The sample size sixty-four was the largest practicable on the ACE computer.

The whole of this routine was performed forty times, using each time a different choice of n-tuples, ith row patterns, paradigms, trial head-patterns, and h value. In Fig. 4(*a*) the numbers of errors per sixty-four trial head-pattern sample after storing one paradigm is plotted against h for the forty runs. Figs. 4(*b*), (*c*) and (*d*) similarly are plots of numbers of errors for the sample of sixty-four trial head-patterns after storing two, three and four paradigms respectively.

Figs. 5(*a*), (*b*), (*c*), (*d*) are plots against h of the numbers of correct paradigm outputs per sixty-four trial head-pattern sample after storing one, two, three and four paradigms, respectively. These plots show how often the outputs from schemes A and B are the same when there is in fact an ith row stored paradigm corresponding to a trial head-pattern. For many trial head-patterns there is no ith row stored paradigm: this, together with occurrences of errors, accounts for the large number of zeros in Figs. 5(*a*)–(*d*).

(It is not surprising that the numbers of correct paradigm outputs for lower h tend to be the greater because

(i) as h decreases the number of patterns which occur more than once in the ith row increases,
(ii) as h increases, the chance of storing a paradigm

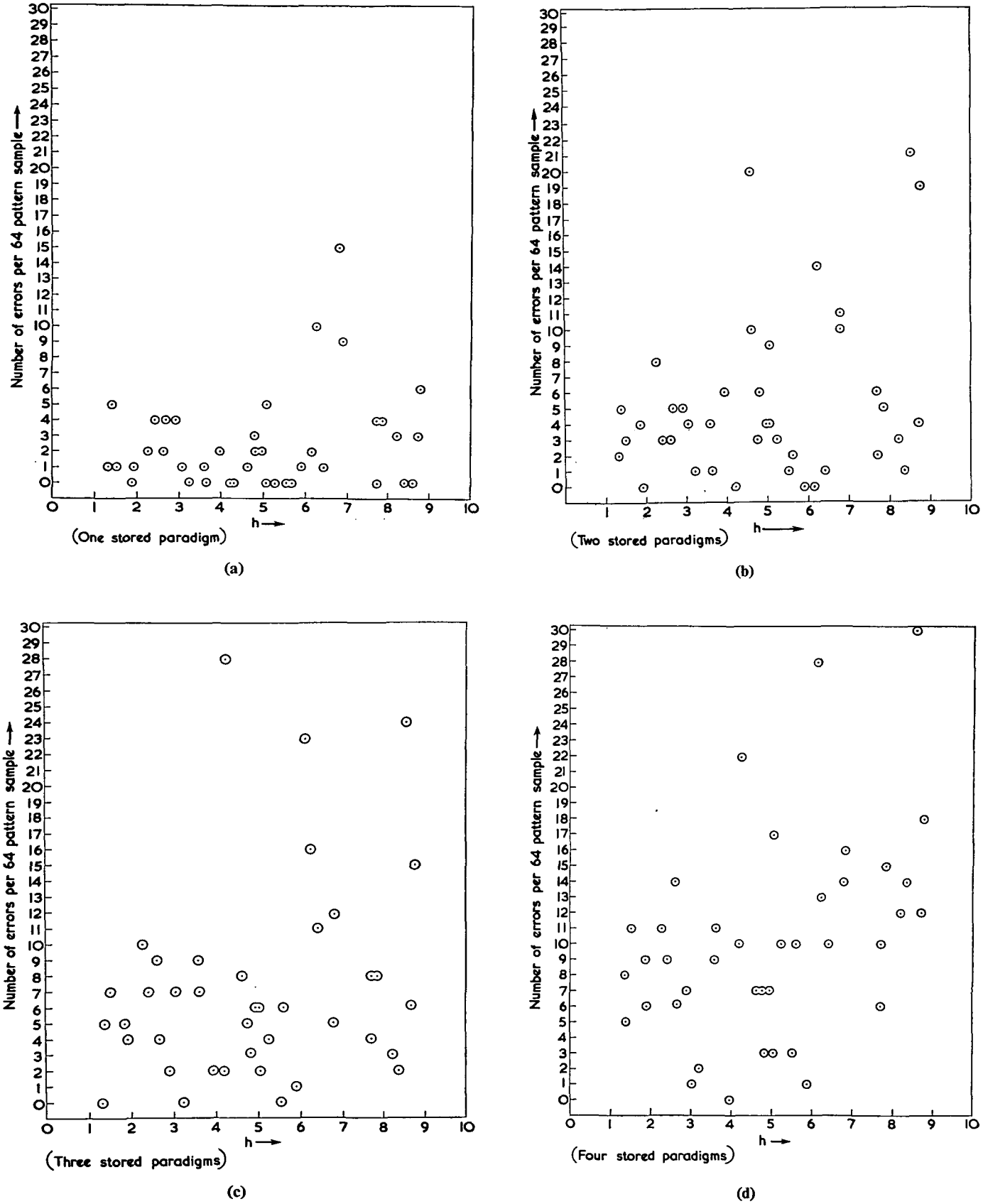

Fig. 4.—Plots of numbers of errors per sample against *h*

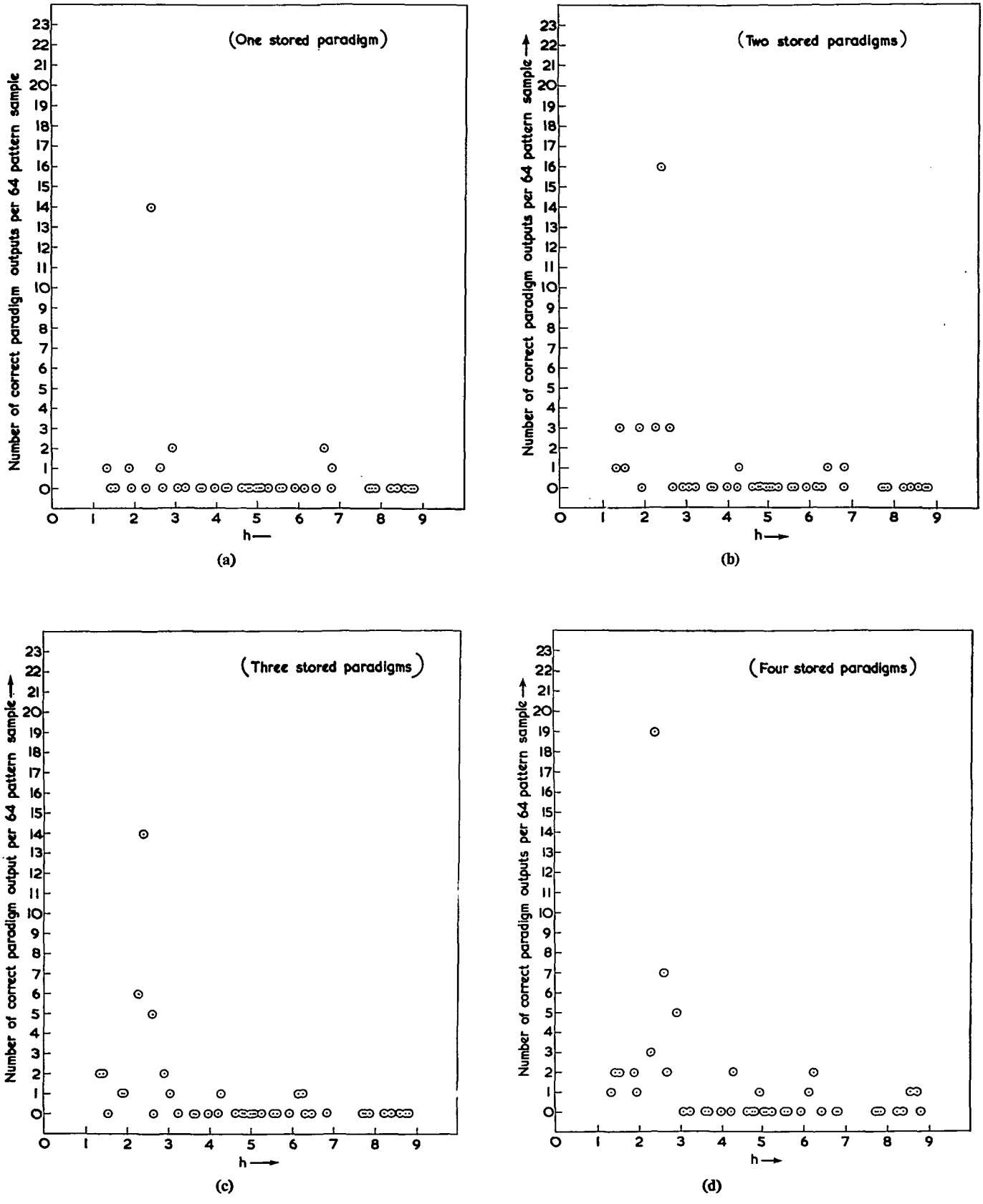

Fig. 5.—Plots of numbers of correct paradigm outputs per sample against *h*

which is not an ith row pattern increases. Such a pattern can, of course, never be a correct paradigm output.)

In Fig. 6, the standard error of the estimated value of h is plotted against the estimated value of h for the forty runs. The performance of scheme B was not explored for $h > 9$, because substituting $N = 10$, $n = 4$ in the condition

$$h < 2^{(N-n)}/N$$

we obtain $h < 6{\cdot}4$.

3. Concluding remarks

(*a*) *Modification of scheme B*

We suggest that the number of errors made by scheme B could be reduced by two developments which it was impracticable to try out on the ACE computer, since the program already took a very long time to run. Without simulation we cannot say how much improvement these two developments would yield.

(i) In constructing, for example, F_{i2} from S_{i2}, we could insist not only that for a bit d to be in state 1 in F_{i2}, the n n-tuples to which d belongs must each have at least one state in S_{i2} in which d is in state 1, but now also insist that these n-tuple states must have occurred *jointly* in a stored paradigm.

(ii) In scheme B an ith row n-tuple state, t, generally corresponds to a number of head-pattern n-tuple states, and for each of these there is a different *token* of t stored in scheme B. If t occurred in a stored paradigm, all tokens of t would become members of Q_i. We now propose the following modification. Let T_0 be the set of head-pattern n-tuple states for which a corresponding ith row n-tuple state is included in a given paradigm which is presented to scheme B for storage. A pattern G_0 is constructed from T using exactly the same rule as was used in constructing F_{i0} from S_{i0}, and the terminal pattern G_w is found similarly. It is now stipulated that only tokens of paradigm pattern n-tuple states for which the corresponding n-tuple state is in G_w become members of Q_i.

(*b*) *Finding the similarity ranking*

Of course the outstanding problem is how, in practice, the correct similarity ranking for patterns could be discovered. One might ask whether it is logically possible for a machine, regardless of size and cost, to discover the ranking; and if so, whether a plausibly economical design could possibly be found for the machine.

By trying out scheme A for all possible rankings, the ranking giving the best performance in character recognition could be found, but this would require a prohibitive number of trials. So we ask whether, having made relatively few trials, a machine could possibly infer the entire ranking. This is such a formidable problem that we think it best to tackle first a series of simpler and clearer problems of this type formulated elsewhere (Ullmann, 1964).

Acknowledgements

The work described above has been carried out as part of the research programme of the National Physical Laboratory, and this paper is published by permission of the Director of the Laboratory.

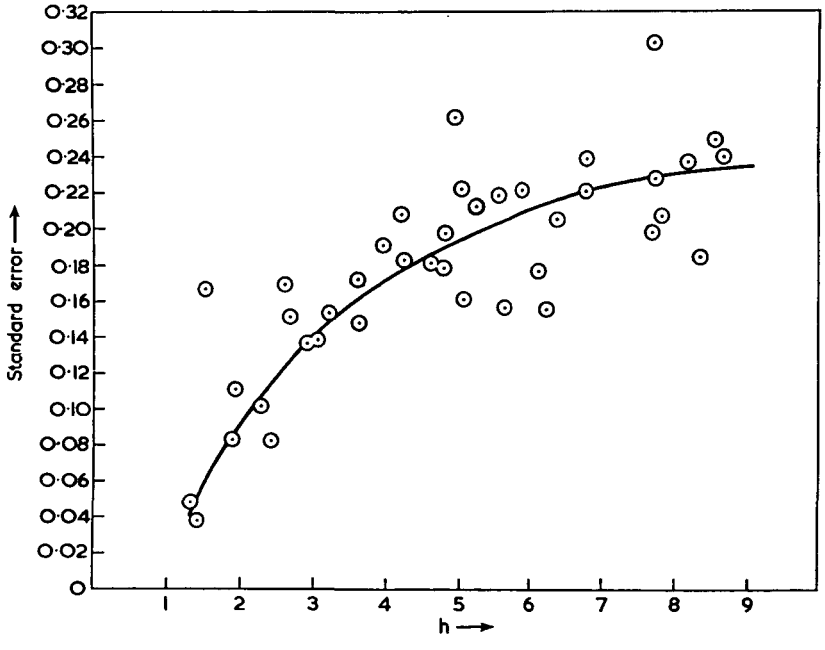

Fig. 6.—Plot of standard error of estimated h against estimated h

References

AIZERMAN, M. A. (1963). "Automatic Control Learning System." Paper presented at Second International Congress of the International Federation of Automatic Control, Basle. Not yet published.

BARUS, C. (1962). "A Scheme for Recognising Patterns from an Unspecified Class." In *Optical Character Recognition* edited by Fischer, Pollock, Raddack and Stephens, Spartan Books, pp. 227–248.

CLOWES, M. B., and PARKS, J. R. (1961). "A New Technique in Automatic Character Recognition," *The Computer Journal*, Vol. 4, pp. 121–128.

GRIMSDALE, R. L., SUMNER, F. H., TUNIS, C. J., and KILBURN, T. (1959). "A System for the Automatic Recognition of Patterns," *Proc. IEE*, Vol. 106, Pt B, No. 26, pp. 210–221.

KAMENTSKY, L. A., and LIU, C. N. (1963). "Computer Automated Design of Multifont Print Recognition Logic," *IBM Journal of Research and Development*, Vol. 7, No. 1, pp. 2–13.

SEBESTYEN, G. S. (1962). *Decision-making Processes in Pattern Recognition*, Macmillan: New York and London.

UHR, L., and VOSSLER, C. (1961). "A Pattern Recognition Program that Generates, Evaluates, and Adjusts its own Operators." Proceedings of the Western Joint Computer Conference, pp. 555–561.

ULLMANN, J. R. (1962). "A Consistency Technique for Pattern Association," *I.R.E. Transactions on Information Theory*, Volume IT-8, No. 5, pp. 74–81.

ULLMANN, J. R. (1964). Article in a *Festschrift* to be published in honour of Norbert Wiener.

Book Review—Automation in Bankwesen (*continued from p.* 274)

3rd Chapter: New types of work
Introduction
1. Planning
   Two sides of the work—systems work—analysis of work to date—construction of the new—programming—detailed work processes—tests.
2. Organization.
3. Executive activities.
4. Programming: a new style of work
   Powers—pretraining—special characteristics—team work—satisfaction—responsibility and efficiency—company position

4th Chapter: Long-term aspects
Preface
1. The position of automation specialists.
2. Possibilities of extending automation bookkeeping—statistics—operations research—the automatic bank.
3. Promoting and restricting factors in development.
4. View of the future.

5th Chapter: Bank automation and the total economy
1. Industrial revolution.
2. Price stabilization.
3. Labour market.
4. Economic position.

FINAL CONSIDERATIONS: Automation—duties and aims

APPENDIX
1. The automatic electronic data processing machine.
   The construction of the electronic computer.
   Central control—input devices—output devices—external storage.
   The co-operation of the various components of the computer.
2. Bibliography.
3. Index.

A. POWER