A PATTERN-RECOGNITION PROGRAM THAT GENERATES, EVALUATES, AND ADJUSTS ITS OWN OPERATORS

by Leonard Uhr & Charles Vossler

Background Review

The typical pattern-recognition program is either elaborately preprogrammed to process specific arrays of input patterns, or else it has been designed as a *tabula rasa,* with certain abilities to adjust its values, or "learn." The first type often cannot identify large classes of patterns that appear only trivially different to the human eye, but that would completely escape the machine's logic (Bailey and Norrie, 1957; Greanias et al., 1957). The best examples of this type are probably capable of being extended to process new classes of patterns (Grimsdale et al., 1959*a*; Sherman, 1959). But each such extension would seem to be an *ad hoc* complication where it should be a simplification, and to represent an additional burden of time and energy on both programmer and computer.

The latter type of self-adjusting program does not, at least as yet, appear to possess methods for accumulating experience that are sufficiently powerful to succeed in interesting cases. The random machines show relatively poor identification ability (Rosenblatt, 1958, 1960*a*). (One exception to this statement appears to be Roberts' modification of Rosenblatt's Perceptron (Roberts, 1960). But this modification appears to make the Perceptron an essentially nonrandom computer.) The most successful of this type of computer, to date, simply accumulates information or probabilities about discrete cells in the input matrix (Baran and Estrin, 1960; Highleyman and Kamentsky, 1960). But this is an unusually weak type of learning (if it should be characterized by that vague epithet at all), and

this type of program is bound to fail as soon as, and to the extent that, patterns are allowed to vary.

Several programs compromise by making use of some of the self-adapting and separate operator processing features of the latter type of program, but with powerful built-in operations of the sort used by the first type (Doyle, 1960; Unger, 1959). They appear to have gained in flexibility in writing and modifying programs; but they have not, as yet, given (published) results that indicate that they are any more powerful than the weaker sort of program (*e.g.*, Baran and Estrin) that uses individual cells in the matrix in ways equivalent to their use of "demons" and "operators." A final example of this mixed type of program is the randomly coupled "*n*-tuple" operator used by Bledsoe and Browning (1959; 1961*a*). In this program, random choice of pairs, quintuples and other tuples of cells in the input matrix is used to compose operators, in an attempt to get around the problems of preanalyzing and preprogramming. This method appears to be guaranteed to have at least as great power as the single cell probability method (Uhr, 1961*b*). But it has not as yet demonstrated this power. And it would, like most of the other programs discussed (or known to the authors) fall down when asked to process patterns which differed very greatly from those with which it had originally "gained experience" by extracting information (Uhr, 1960).

Summary of Program Operation

In summary, the original running pattern recognition program works as follows: Unknown patterns are presented to the computer in discrete form, as a 20×20 matrix of zeros and ones. The program generates and composes operators by one of several random methods, and uses this set of operators to transform the unknown input matrix into a list of characteristics. Or, alternately, the programmer can specify a set of pregenerated operators in which he is interested.

These characteristics are then compared with lists of characteristics in memory, one for each type of pattern previously processed. As a result of similarity tests, the name of the list most similar to the list of characteristics just computed is chosen as the name of the input pattern. The characteristics are then examined by the program and, depending on whether they individually contributed to success or failure in identifying the input, amplifiers for each of these characteristics are then turned up or down. This adjustment of amplifiers leads eventually to discarding operators which produce poor characteristics, as indicated by low amplifier settings, and to their replacement by newly generated operators.

One mode of operation of the present program is to begin with no operators at all. In this case operators are initially generated by the pro-

gram at a fixed rate until some maximum number of operators is reached. The continual replacement of poor operators by new ones then tends to produce an optimum set of operators for processing the given array of inputs.

Details of Program Operation

The program can be run in a number of ways, and we will present results for some of these. The details of the operation of the program follow.

1. An unknown pattern to be identified is digitized into a 20×20 0–1 input matrix (Fig. 1).
2. A rectangular mask is drawn around the input (its sides defined by the leftmost, rightmost, bottommost, and topmost filled cells) (Fig. 2).
3. The input pattern is transformed into four 3-bit characteristics by each of a set of 5×5 matrix operators, each cell of which may be visualized as containing either a 0, 1, or blank. These small matrices which measure local characteristics of the pattern are translated, one at a time, across and then down that part of the matrix which lies within the mask. The operator is considered to match the input matrix whenever the 0's and 1's in the operator correspond to identical values in the pattern, and for each match the location of the center cell of the 5×5 matrix operator is temporarily recorded (Fig. 3). This information is then summarized and scaled from 0 to 7 to form four 3-bit characteristics for the operator. These represent (1) the number of matches, (2) the average horizontal position of the matches within the rectangular mask, (3) the average

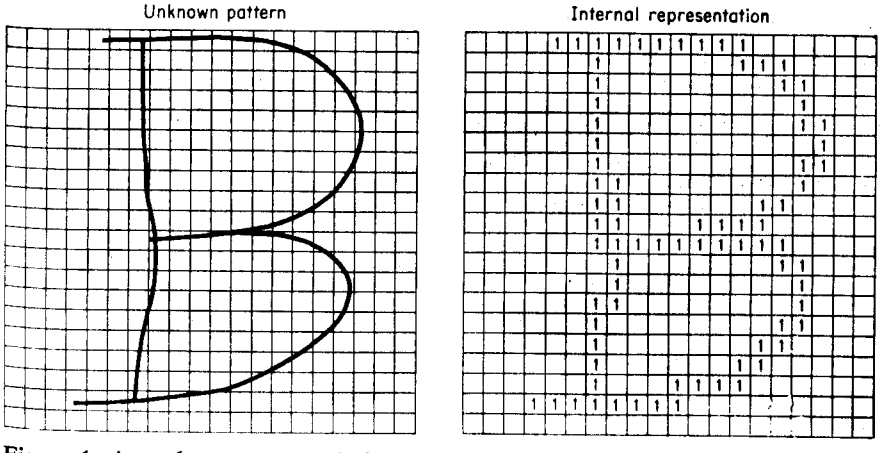


Figure 1. An unknown pattern is input as a 20×20 matrix with the cells covered by the pattern represented by "1's" and the other cells by "0's."

vertical position of the matches, and (4) the average value of the square of the radial distance from the center of the mask.

A variable number of operators can be used in any machine run. This can mean either a number preset for that specific run, or a number that begins at zero and expands, under one of the rules described below, up to the maximum. The string of 25 numbers which defines a 5×5 matrix operator can be generated in any of the following ways (Fig. 4):

a. A preprogrammed string can be fed in by the experimenter.
b. A random string can be generated; this string can be restricted as to the number of "ones" it will contain, and as to whether these "ones" must be connected in the 5×5 matrix. (We have not actually tested this method as yet.)
c. A random string can be "extracted" from the present input matrix and modified by the following procedure (which in effect is imitating a certain part of the matrix). The process of inserting blanks

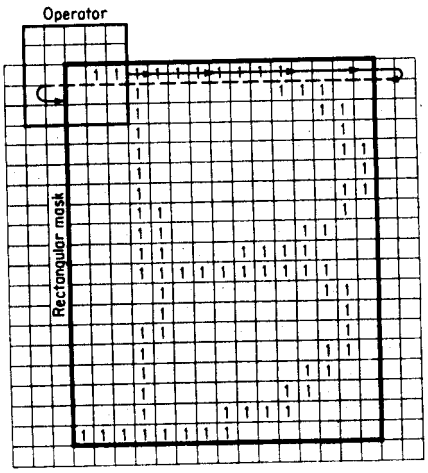
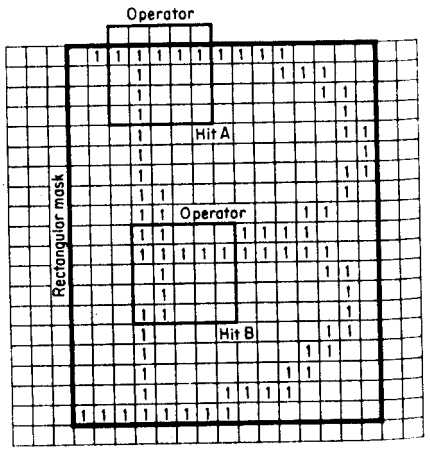
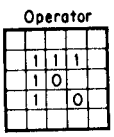


Figure 2. A rectangular mask is drawn around the unknown pattern. Each of the 5×5 matrix "operators" is then translated over the pattern.





Hit	X	Y	R^2
A	1	0	4
B	2	4	0
N = 2	2	2	2

Figure 3. The operator at the lower left in the figure is shown in the two positions where it matches the input matrix. An operator gives a positive output each time its "1's" cover "1's" and its "0's" cover "0's" in the unknown pattern.

in the extracted operator allows for minor distortions in the local characteristics which the operator matches.

(1) A 5×5 matrix is extracted from a random position in the input matrix.

(2) All "zero" cells connected to "one" cells are then replaced by blanks.

(3) Each of the remaining cells, both "zeros" and "ones," is then replaced by a blank with a probability of $\frac{1}{2}$.

(4) Tests are made to ensure that the operator does not have "ones" in the same cells as any other currently used operator or any operator in a list of those recently rejected by the program. If the operator is similar to one of these in this

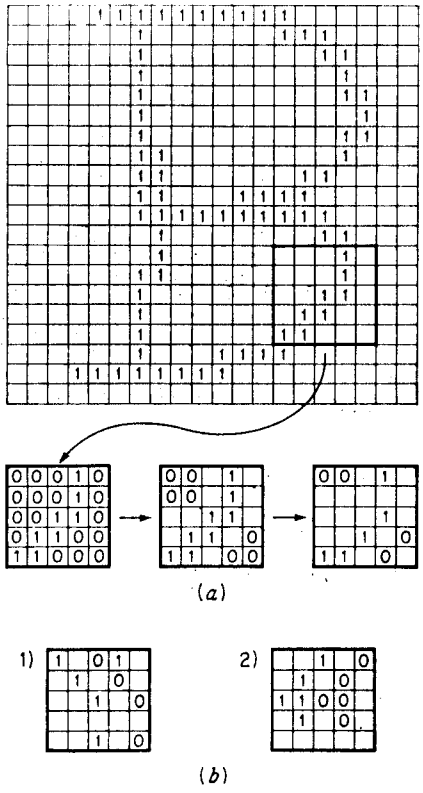


Figure 4. Operators are generated within the 5×5 matrix by either: (*a*) extraction from the input pattern (random placement of a 5×5 matrix, elimination of "0's" connected to "1's" and elimination of each of the remaining cells with a probability of $\frac{1}{2}$) or (*b*) by random designation of cells as either "0" or "1" (choose a "1," then place a "0" two cells to its right). In 1) from 3 to 7 "1's" are chosen completely at random, while in 2) the choice is limited to connected cells.

respect a new operator is generated by starting over at step 1 (Fig. 5).

4. A second type of operator is also used. This is a combinatorial operator which specifies one of 16 possible logical or arithmetic operations and two previously calculated characteristics which are to be combined to produce a third characteristic. These operators are generated by the program by randomly choosing one of the possible operations and the two characteristics which are to be combined. This random generation process is improved by generating a set of ten operators, and then pretesting these using the last two examples of each pattern which have been saved in memory for this purpose. This pretesting is designed to choose an operator from the set which produces characteristics that tend to be invariant over examples of the same pattern yet vary between different patterns.

Since these operators may act upon characteristics produced by previous operators of the same type, functions of considerable complexity may be built up.

5. The two types of operators just described produce a list of characteristics by which the program attempts to recognize the unknown input

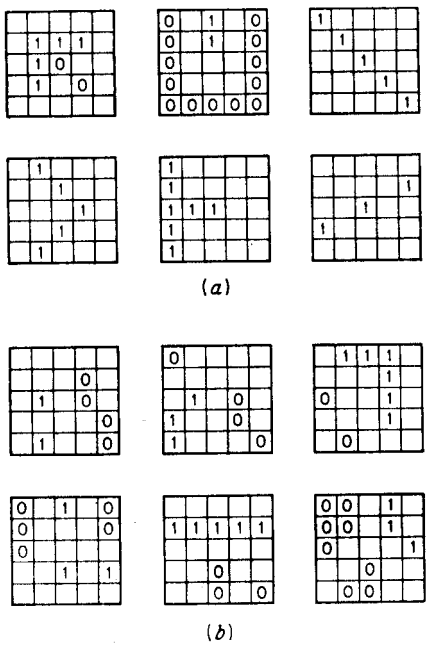


Figure 5. (*a*) Some typical examples of preprogrammed operators are shown. (*b*) Six of the operators generated by the program, during a run that reached 94 per cent success on 7 sets of 5 patterns, are shown.

pattern (Fig. 6). At any time the program has stored in memory a similar list of characteristics for each type of pattern which the program has previously encountered. Corresponding to each list of characteristics in memory is a list of 3-bit amplifiers, which gives the current weighting for each characteristic as a number from 0 to 7.

The recognition process proceeds by taking the difference between each of the characteristics for the input pattern and those in the recognition list of the first pattern. These differences are then weighted by the corresponding pattern amplifiers, and then by general amplifiers which represent the average of the pattern amplifiers across all patterns, producing a weighted average difference between the input list and the list in memory. This average difference is multiplied by a final "average difference" amplifier to obtain a "difference score" for the list in memory. When a difference score has been computed for each list in memory, the name of the list with the smallest score is printed as the name of the input pattern (Fig. 7).

6. After each pattern is recognized the program modifies pattern amplifiers in those patterns which have difference scores less than or only slightly above the difference score for the correct pattern (Fig. 8). This means that the program will tend to concentrate on the difficult discrimination problems, since amplifiers are adjusted only in those patterns which appear similar to the correct pattern in terms of the difference scores and therefore make identification of the input difficult. The correct pattern is compared with each of the similar patterns in turn. Each characteristic in the memory lists for a pair of patterns is examined individually, and a determination is made as to whether the correct pattern would have been chosen if the choice had been made on the basis of this characteristic alone. If this one characteristic would have identified the correct pattern, then the corresponding amplifier is turned up by one. If it would have identified the wrong pattern then the amplifier is turned down by one. If no information is given by the characteristic, for example, if it is the same

	← Matrix operators →									← Combinatorial operators →		
Pattern	Operator 1				Operator 2					Characteristic		
Name	N	X	Y	R^2	N	X	Y	R^2	...	m-2	m-1	m
?	2	2	2	2	2	3	1	6	....	6	7	4
A	3	3	4	1	4	0	1	1	...	1	2	3
B	2	2	3	2	3	3	2	5	...	4	7	5
C	4	5	6	5	1	0	0	4	...	2	1	7

Figure 6. Operator outputs are listed for the unknown pattern in the same format as in lists stored in memory.

for both patterns, then the amplifier is turned down with a probability of $\frac{1}{8}$. If the pattern compared with the correct pattern had the higher difference score then the amplifiers are adjusted only in that pattern. Otherwise, amplifiers are adjusted in both patterns. This means that if several patterns obtained lower scores than the correct pattern then the amplifiers in the correct pattern will be drastically changed, since they will change when compared with each of these patterns.

The list of characteristics in memory for the pattern just processed is then modified. The first time a pattern is encountered its list of computed characteristics is simply stored in memory along with its name. On the second encounter of a pattern each of the characteristics in memory is replaced by the new characteristic with a probability of $\frac{1}{2}$. For the third and following encounters each characteristic is replaced by the new value with a probability of $\frac{1}{4}$. Since about $\frac{1}{4}$ of the characteristics will be changing each time, after several examples of a pattern have been processed, the list of characteristics in memory will tend to be more similar to the characteristics of the last patterns processed than to those processed

PATTERN *A*

Characteristics (A)	:	3	3	4	1	4 . . .	3
Input (?)	:	2	2	2	2	2 . . .	4
Difference |A−?|	:	1	1	2	1	2 . . .	1
Pattern amplifiers	:	2	3	1	2	0 . . .	3
General amplifiers	:	3	3	1	1	0 . . .	3
Diff. X amplifiers	:	6	9	2	2	0 . . .	9

Weighted av. diff.	Av. diff. amplifier	Diff. score
$\frac{28}{27} = 1.04$	61	63

PATTERN *B*

Characteristics (B)	:	2	2	3	2	3 . . .	5
Input (?)	:	2	2	2	2	2 . . .	4
Difference |B−?|	:	0	0	1	0	1 . . .	1
Pattern amplifiers	:	4	3	2	3	2 . . .	2
General amplifiers	:	3	3	1	1	0 . . .	3
Diff. X amplifiers	:	0	0	2	0	0 . . .	6

Weighted av. diff.	Av. diff. amplifier	Diff. score
$\frac{8}{32} = 0.25$	60	15

Figure 7. Differences are obtained between the characteristics for the input pattern and each list of characteristics in memory. These differences are then weighted by the product of the "general amplifiers" and "pattern amplifiers," giving a weighted average difference for each list in memory. When multiplied by corresponding "average difference amplifiers," the weighted average differences give "difference scores" for each pattern in memory. The name of the pattern with the smallest "difference score" is chosen as the name of the input.

earlier. However, to the extent that the learning process is able to produce operators giving invariant characteristics for a single pattern, the list of characteristics will be representative of all the examples processed. The reason for not simply using the average value for each characteristic is that this would require saving in memory more than the 3 bits otherwise needed for each characteristic, as well as saving an indication of the number of times each characteristic had been calculated for each pattern.

An alternate scheme which we tried involved saving the highest and lowest values obtained by each characteristic, and averaging these to obtain a mean value with which to compare the input. This worked quite well in all our test runs, which used a few samples of each pattern. But there is the possibility that with large numbers of examples of a pattern, all the characteristics will eventually have very large ranges; that is, the lower bounds will tend to be 0 and the upper bounds will tend to be 7.

7. The average difference amplifiers which are used in the final step of the recognition process provide only coarse adjustments. These amplifiers are initially set to some fixed value, *e.g.*, 60, and are then adjusted for the same pairs of patterns as the pattern amplifiers. The amplifier for the correct pattern is turned down by N if there are N incorrect patterns, and the amplifier for each of the similar patterns is turned up by one.

8. The general characteristic amplifiers are now computed by averaging the pattern amplifiers across all patterns. These indicate the general value of each characteristic in the recognition process and form the basis for the construction of success counts which control the replacement of operators. Since the combinatorial operators combine characteristics to produce other characteristics, the success count should reflect both the value of a charac-

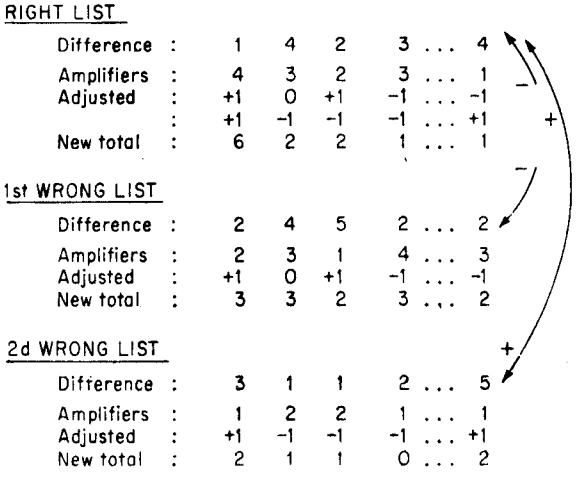


Figure 8. The pattern amplifiers for certain lists are adjusted to *increase* weightings of individual characteristics that gave differences in the right direction, and to *decrease* weightings that gave differences in the wrong direction.

teristic in the recognition process and the importance of this characteristic in aiding the creation of other, possibly important characteristics.

9. This success count is formed by first storing the value of the general characteristic amplifier corresponding to each characteristic in a table for success counts. Then starting with the last combinatorial operator and working back through the list of these operators, ½ the value of the success count for the characteristic corresponding to the operator is added to the success counts of the two characteristics which the operator combines. Finally, two times the general characteristic amplifier setting is added to each success count.

10. Whenever a new operator is generated, the characteristics produced by the operator are computed for each of the possible patterns using the last example of each pattern, which has been saved in the computer memory. These newly calculated characteristics are then inserted into the list of characteristics for their respective patterns. At the same time the pattern amplifier settings for each of these new characteristics are set to 1 so that the characteristic will have very little weight in computing a difference score until it has been turned up as a function of proved ability at differentiation. Since the general amplifier for a characteristic is simply the average of the pattern amplifiers, it will also be 1 for the new characteristic. The success count of a new characteristic which is not combined to produce other characteristics is then 3 and this value will tend to increase if the operator proves to be valuable. On the other hand if a success count drops below 3 (or in the case of a matrix operator, if the average value of the success counts of its four characteristics drops below 3) the operator is rejected and a new operator is generated to take its place.

The pattern amplifiers play a crucial part both by aiding directly in the recognition process and by providing the information which ultimately determines the generation of new operators to replace poor ones. Since the adjustment of these amplifiers is made selectively, based on their individual success or failure in distinguishing pairs of patterns where confusion is likely, the operators rejected by the program will tend to be those which are not useful in making the more difficult discrimination. Also, because amplifiers are usually changed more drastically when the computer makes an incorrect guess, the 5×5 matrix operators will have a higher probability of being extracted from unrecognized patterns. Although the rules governing the learning process seem rather arbitrary in many cases, and it is difficult to describe their effects quantitatively, qualitative effects, such as this ability to concentrate on difficult problems, are fairly easy to show. The description of the program's operation shows that the emphasis is not so much on the design of a specific problem-solving code as it is on the design of a program which, at least in part, will construct such a problem-solving code as a result of experience.

It is interesting to note that the memory of the program exists in at least three different places: (1) in the lists of characteristics in memory, (2) in the settings of the various amplifiers, and (3) in the set of operators in use by the program. While the lists of characteristics bear some direct relationship to the individual patterns processed by the program, the values of the amplifiers and the set of operators in use by the program depend in a more complex way on the whole set of patterns processed by the program, and on the program's success or failure in recognizing these patterns. The learning in the first case, which involves simply storing characteristics in memory, is merely "memorization" or "learning by rote." In the second case, the learning is more subtle for it involves the program's own analysis of its ability to deal with its environment, and its attempt to improve this ability.

Test Results of Original Program

The original program was written for the IBM 709 and required about 2000 machine instructions. The time required to process a single character was about 25 seconds when 5 different patterns were used and 40 seconds when each character had to be compared with ten possible patterns in memory. While such times are not excessive, they are large enough to make it impractical to run extremely large test cases.

In several early runs which we made, 48 preprogrammed matrix operators were used. These were designed to measure such things as straight and curved lines, the ends of vertical and horizontal strokes, and various other features. The program was tested using seven different sets of the five hand-printed characters A, B, C, D, and E. These involved a fair amount of distortion, and variation in size, but were not rotated to any great extent.

The program's performance on the last three or four sets in a run varied from about 70% to 80% depending on various changes which were made to the rules governing the learning process.

One run was made using the individual cells of the input matrix as first level operators, building up higher-level combinatorial operators on these. This gave little better than 30% success. Finally, this version of the program was tested without any preprogrammed operators, the program achieved 97% on known and 70% on unknown examples of a ten-letter alphabet. It also showed ability to recognize simple drawings of objects.

Test Results of Revised Program

The original program was modified, chiefly to increase its running speed, and secondarily to simplify some of its logic. In order to make use of logical machine instructions on the 709, all characteristics and their ampli-

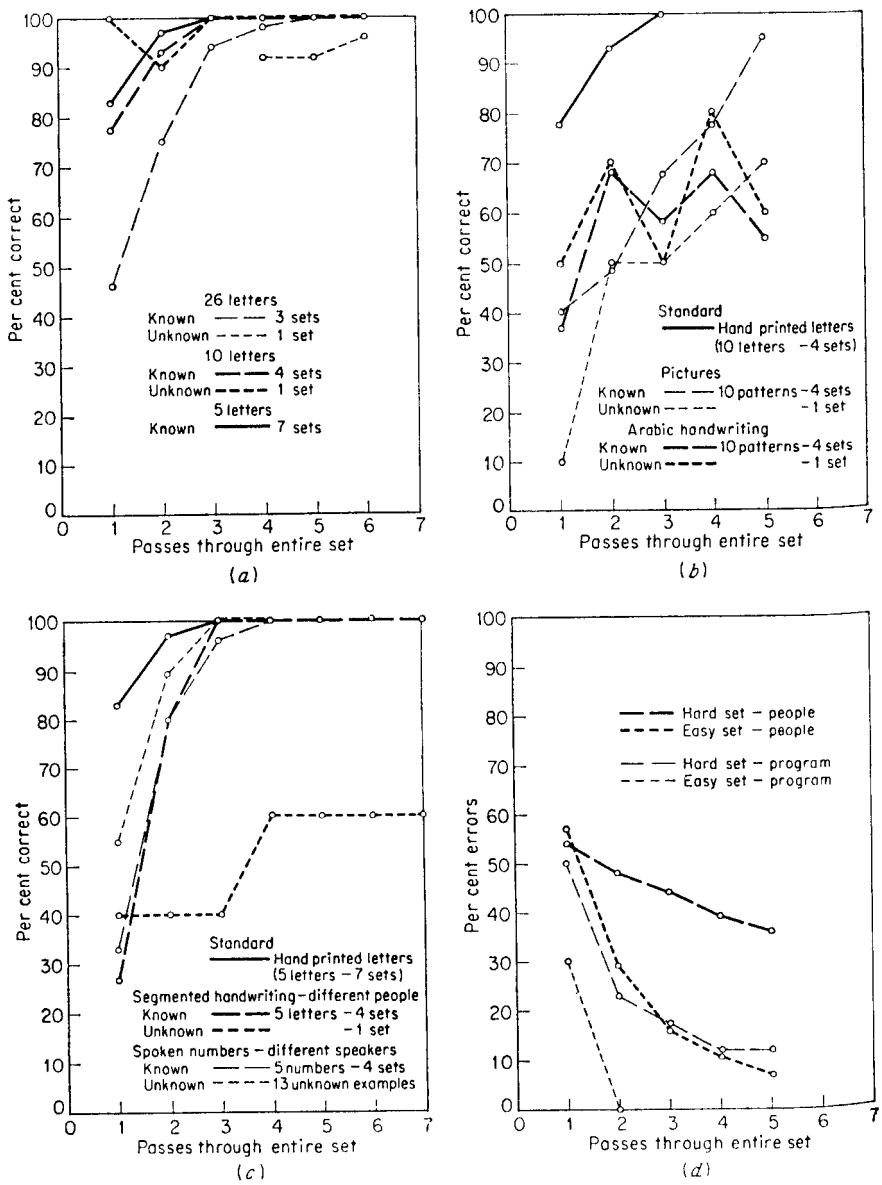


Figure 9. (*a*) Results of the computer simulation program. Hand-printed alphabetic patterns. Per cent correct on several sets of a 26-pattern, a 10-pattern, and a 5-pattern array. The program was tested on both known and unknown sets of patterns. (*b*) Results with two additional 10-pattern arrays: (1) line drawings of pictures (different examples of each of 5 faces and 5 objects), (2) arabic handwriting (written by the same person). The program was tested on both known

fiers were reduced to 1-bit values. The revised program stores nine one-bit values for each operator—whether it hit at least once in each of nine parts of the matrix. Operators are weighted either 1 (to be used) or 0 (not to be used), referring to the characteristics they give for each pattern stored in memory. Operators are eliminated when they have given wrong outputs on the last n example for which the program as a whole has been wrong. The "general amplifiers" have been eliminated. These changes effected an increase of speed by a factor of about 40. Thus this program takes about 1 second per example for a 10-pattern alphabet on the 709, and less than .2 second on the 7090. This program is probably weaker than the original program, since it has virtually no range within which to search for a good set of weightings for its operators. But its increased speed led to its use for the bulk of our tests on this version of the program.

The speeded up program has been tested on several different types of input patterns, as shown in Fig. 9. In most cases, results were quite similar on both "known" examples (that is, examples the program had previously processed and hence had learned from) and "unknown" examples (that is, different from the ones used in learning, and also produced by different people). Figure 9*a* shows results for several different sizes of pattern arrays, all of hand-printed capital letters, printed by different people. These results show relatively little decrease in the program's abilities as the array size is increased, at least up to the 26-letter alphabet. Thus, on the sixth pass through the 26-letter alphabet the program was 100% correct on known patterns and 96% correct on unknown patterns.

Figure 9*b* presents results for two 10-pattern arrays. These were: (1) line drawings of cartoon faces and simple objects (such as shoes and pliers), each copied from a different picture, as found in cartoon strips and mail-order catalogs (Fig. 10 presents some examples of the cartoons), and (2) handwritten arabic letters, written by the same person. The program achieved 95% success on known and 70% success on unknown pictures, and 60% success on known and 55% success on unknown arabic letters (segmented handwriting) in the fifth pass. Figure 9*c* presents results from two 5-pattern arrays: (1) digitized and degraded sound spectrograms of speech (the numbers "zero," "one," "two," "three," and "four," as spoken by different speakers) (Uhr and Vossler, 1961*d*), and (2) segmented lower-case handwriting, written by different people. The

---

and unknown sets of patterns. (*c*) Results with two additional 5-pattern arrays: (i) spoken numerals (spoken by different people), (ii) segmented handwriting (written by different people). The program was tested on both known and unknown sets of patterns. (*d*) Results from a comparison experiment. Per cent errors for the program and mean per cent errors for human subjects (from 6 to 10 subjects per point) on one hard and one easy set of "meaningless" patterns. Both sets contained five variant examples of each of five patterns.

program achieved 100% success on both known and unknown spoken numerals by the fourth pass, and 100% success on known handwriting by the third pass. It achieved 60% success on the unknown handwriting, but it is likely that it would have improved further on these inputs if it had been given more opportunity to learn (once it achieves 100% success on known patterns it does not benefit appreciably from subsequent learning experiences).

Finally, the program's performance was compared with the performance of human subjects on sets of "meaningless" patterns. This sort of pattern minimizes the effects of the human being's lifetime of experience and resulting associative context. Figure 9*d* presents results from two such experiments, in both of which the program performed appreciably better than did any of the human subjects. Three additional experiments previously reported in the psychological literature were replicated. In all cases the program performed at a higher level than did the human subjects (Uhr, Vossler, and Uleman, 1962).

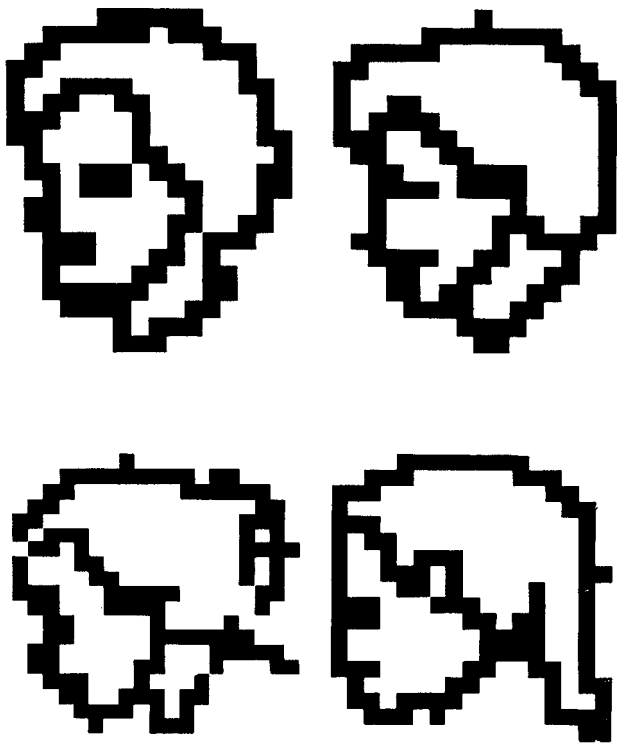


Figure 10. Two examples of each of two cartoon faces as presented to the simulation program (digitized by hand into a 20×20 matrix, after optical projection from the newspaper original).

This program was also tested for its ability to handle continuous patterns such as handwritten words and spoken sentences. Simple additional subroutines were written to allow it to input matrices any number of columns long, to make very primitive tentative segmentations (in every *n*th column, *n* usually around 7, and in columns with fewer than two filled cells), and to decide among the various alternatives at the various different tentative segmentation points. The program reached 100% correct performance when asked to segment and recognize the words, or alternatively the phonemes, in the simple sentence "Did Dad say before," spoken by different people.

On the handwritten sequences "pattern one," "pattern two," "pattern three" (written by different people), the program reached about 60% success in recognizing the letters. These tests do not in any adequate way sample the range of problems to be encountered with such stimuli. But they give some indication that the program is capable of at least beginning to handle continuous inputs. And it should be relatively easy to improve upon this performance by adding more sophisticated segmenting methods and a straightforward method (such as the use of letter *n*-tuple frequencies in the language) for making use of contextual information.

Discussion

When this program is given a neurophysiological interpretation, or a neural, net analog, it can be seen to embody relatively weak, plausible, and "natural-looking" assumptions. The 5×5 matrix operator is equivalent to a 5×5 net of input retinal cones or photocells converging on a single output, with "ones" denoting excitatory and "zeros" denoting inhibitory connections, and the threshold for firing the output unit set at the sum of the "ones." Each translation step of the operator matrix over the larger matrix gives a sequential simulation of the parallel placement of many of these simple neural net operators throughout the matrix. Each different operator, then, is the equivalent of an additional connection pattern between input cones, firing onto a new output unit that computes the output for that operation. This is all quite plausible for the retina as known anatomically, with a single matrix of cones in parallel that feed into several layers of neurons. Evidence for excitatory and inhibitory connections is also strong (Hartline, 1938). And there is even beginning to be evidence of several types of simple net operators that exist in parallel iterated form throughout the retinal matrix [four of these as determined by Lettvin, Maturana, McCulloch, and Pitts in the frog (1959); and probably even more as determined by Hubel and Wiesel in the cat (1959)].

It would seem, however, that the known physiological constraints and the plausible geometric constraints on operators would suggest fewer than

the 40-odd operators that we have used [or than the 30-odd used by Doyle (1960) or the 75 used by Bledsoe and Browning (1959) ignoring the fact that they cannot be so easily interpreted neurophysiologically]. For example, straight-line and sharp-curve operators would seem to be more plausible in terms of the ease of connection and the importance of the information to which they respond. A possible operator that might overcome this problem, with which we are now working, is a simple differencing operator that will, by means of several additional layers of operations, first delineate contour and then compute successively higher order differences, and hence straightness, slope and curvature, for the unknown pattern. This operator appears to be equivalent to a simple net of excitatory and inhibitory elements (Uhr and Vossler, 1961*a*).

This, then, suggests that the mapping part of the program would be effected by two layers of parallel basic units in a neuron netlike arrangement. The matching part might similarly be performed by storing the previously mapped lists in a parallel memory and sweeping the input list, now mapped into the same standard format, through these lists. Finally, the amplifiers can be interpreted as threshold values as to when the differences thus computed lead to an output. The specific pattern characteristic amplifier would be an additional single unit layer lying right behind the memory list; the interpretation of the general amplifiers might be made in terms of chemical gradients, but is more obscure.

Thus a suitable parallel computer would perform all of the operations of this program in from three to ten serial steps. This is a somewhat greater depth than those programs, such as Selfridge's (1959) and Rosenblatt's (1958), that attempt to remain true to this aspect of the visual nervous system. But it is well within the limits, and actually closer to the specifications, of that system. It also takes into consideration the very precise (and amazing) point-to-point and nearness relations that are seen in the visual system, both between several spots on the retina or any particular neural layer, and from retina to cortex (Sperry, 1951). It also is using operators that seem more plausible in terms of neural interconnections—again, in the living system, heavily biased toward nearness.

The size of the over-all input matrix has also been chosen with the requirements of pattern perception in mind. Good psychophysical data show clearly that when patterns of the complexity of alphanumeric letters are presented to the human eye, recognition is just as sure and quick no matter how small the retinal cone mosaic, until the pattern subtends a mosaic of about the 20×20 size, at which time recognition begins to fall off, in both speed and accuracy, until a 10×10 mosaic is reached, at which point the pattern cannot be resolved at all. This further suggests something about the size of the basic operator, when we consider that most letters are composed of loops and strokes that are on the order of ½ or ¼ of the

whole. For our present purposes, the advantage of the 5×5 operator was not only its plausibility but also the fact that it cuts down to a workable size the space within which to generate random operators of the sort we are using when we permute through all possible combinations of the matrix. Again, with the constraint that these random operators be connected, it becomes a more powerful geometry- and topology-sensitive operator, and also a simulation of a more plausible neural net.

Finally, psychophysical evidence also strongly suggests that the resolving power of the human perceptual mechanism is on the order of only two or three bits worth of differentiation as to dimensions of pattern characteristics—things such as length, slope, and curvature (Alluisi, 1957; Miller, 1956*b*; Uhr, 1959*b*). This, again, suggests a 5×5 matrix as a minimum matrix that is capable of making these resolutions.

The specifications for and methods used by living systems, and especially the human visual system, suggest certain design possibilities for a pattern-recognition computer; but they certainly do not suggest the only possibilities. Nor should they be slavishly imitated. They should, however, be examined seriously, for the living pattern recognizers are the only successful systems that we know of today. Nor does it seem that the sort of use we have been making of these human specifications will impose any fundamental limitation on a program such as this, one that generates and adjusts its own operators. We have, in fact, already found the program making a different, and, apparently, more powerful, choice of operators than the choice suggested to us by the psychophysiological data and conjectures we have just described. The program's "learning" methods can now depend both on built-in connections (maturation) and on the inputs that need to be learned. The program will develop differently as a function of different input sets. It appears to be capable of extracting and successfully using information from these sets. This would seem to be as completely adaptive—being adaptive to inputs—as a computer or organism can be expected to be.

This sort of design would seem to have some applicability to a variety of more "intelligent" machines. The program replaces the programmer-analyst by a programmed operator that first generates operators that make effective enough use of the unknown input space, and then makes use of feedback as to the success of these new operators in mapping unknown inputs in order to increase their effectiveness. Thus neither programmer nor program needs to know anything specific about the problem ahead of time. The program performs, as part of its natural routine, the data collection, analysis, and inference that is typically left to the programmer. This would be a foolish waste of time for a problem that had already been analyzed. But pattern recognition, and many other problems of machine intelligence, have not been sufficiently analyzed. The different pattern-

recognition programs are, themselves, attempts to make this analysis. As long as pattern recognition remains in the experimental stage (as it must do until it is effectively solved), a program of this sort would seem to be the most convenient and flexible format for running what is, in effect, a continuing series of experiments upon whose results continuing modifications of theories are made. This becomes an extremely interesting process for the biologist or psychologist, especially to the extent that the program can be interpreted either physiologically or functionally, or at the least does not violate any known data. For the experimentation and concomitant theory building and modification being undertaken today is rapidly building what appears to us to be the first relatively firm and meaningful theoretical structure—for pattern, or form, perception—for the science of "higher mental processes."

Self-generation of operators, by the various methods employed in this program, may also suggest approaches toward solving a wide variety of pattern-recognition and pattern-extraction problems. Thus there is some hope that relatively powerful operators are being extracted and generated as a result of experience with and feedback from the program's quasi-experimental analysis on the body of data that is available to it—its inputs and the consequences of its actions. Further, the level of power of these operators, and the serial ordering of operators can also be placed under similar control. Thus operators need not be overly simple or random to be machine-chosen; nor preprogrammed to be powerful. Rather, they can arise from the problem, and thus be sensitive to the problem, and to changes in the problem.