PATTERN RECOGNITION BY MACHINE

by Oliver G. Selfridge & Ulric Neisser

Can a machine think? The answer to this old chestnut is certainly "yes": Computers have been made to play chess and checkers, to prove theorems, to solve intricate problems of strategy. Yet the intelligence implied by such activities has an elusive, unnatural quality. It is not based on any orderly development of cognitive skills. In particular, the machines are not well equipped to select from their environment the things, or the relations, they are going to think about.

In this they are sharply distinguished from intelligent living organisms. Every child learns to analyze speech into meaningful patterns long before he can prove any propositions. Computers can find proofs, but they cannot understand the simplest spoken instructions. Even the earliest computers could do arithmetic superbly, but only very recently have they begun to read the written digits that a child recognizes before he learns to add them. Understanding speech and reading print are examples of a basic intellectual skill that can variously be called cognition, abstraction or perception; perhaps the best general term for it is pattern recognition.

Except for their inability to recognize patterns, machines (or, more accurately, the programs that tell machines what to do) have now met most of the classic criteria of intelligence that skeptics have proposed. They *can* outperform their designers: The checker-playing program devised by Arthur L. Samuel of International Business Machines Corporation (1959*a*) usually beats him. They *are* original: The "Logic Theorist," a creation of a group from the Carnegie Institute of Technology and the RAND Corporation [Newell, Simon, and Shaw (1956*a*, 1957*b*)] has found proofs for many of the theorems in *Principia Mathematica,* the

monumental work in mathematical logic by A. N. Whitehead and Bertrand Russell (1940). At least one proof is more elegant than the Whitehead-Russell version.

Sensible as they are, the machines are not perceptive. The information they receive must be fed to them one "bit" (a contraction of "binary digit," denoting a unit of information) at a time, up to perhaps millions of bits. Computers do not organize or classify the material in any very subtle or generally applicable way. They perform only highly specialized operations on carefully prepared inputs.

In contrast, a man is continuously exposed to a welter of data from his senses, and abstracts from it the patterns relevant to his activity at the moment. His ability to solve problems, prove theorems and generally run his life depends on this type of perception. We suspect that until programs to perceive patterns can be developed, achievements in mechanical problem-solving will remain isolated technical triumphs.

Developing pattern-recognition programs has proved rather difficult. One reason for the difficulty lies in the nature of the task. A man who abstracts a pattern from a complex of stimuli has essentially classified the possible inputs. But very often the basis of classification is unknown, even to himself; it is too complex to be specified explicitly. Asked to define a pattern, the man does so by example; as a logician might say, ostensively. This letter is A, that person is mother, these speech sounds are a request to pass the salt. The important patterns are defined by experience. Every human being acquires his pattern classes by adapting to a social or environmental consensus—in short, by learning.

In company with workers at various institutions our group at the Lincoln Laboratory of the Massachusetts Institute of Technology has been working on mechanical recognition of patterns. Thus far only a few simple cases have been tackled. We shall discuss two examples. The first one is MAUDE (for Morse Automatic Decoder), a program for translating, or rather transliterating, hand-sent Morse code. This program was developed at the Lincoln Laboratory by a group of workers under the direction of Bernard Gold.

If telegraphers sent ideal Morse, recognition would be easy. The keyings, or "marks," for dashes would be exactly three times as long as the marks for dots; spaces separating the marks within a letter or other character (mark spaces) would be as long as dots; spaces between characters (character spaces), three times as long; spaces separating words (word spaces), seven times as long. Unfortunately human operators do not transmit these ideal intervals. A machine that processed a signal on the assumption that they do would perform very poorly indeed. In an actual message the distinction between dots and dashes is far from clear. There is a great deal of variation among the dots and dashes, and also among

the three kinds of space. In fact, when a long message sent by a single operator is analyzed, it frequently turns out that some dots are longer than some dashes, and that some mark spaces are longer than some character spaces. (See Fig. 1.)

With a little practice in receiving code, the average person has no trouble with these irregularities. The patterns of the letters are defined for him in terms of the continuing consensus of experience, and he adapts to them as he listens. Soon he does not hear dots and dashes at all, but perceives the characters as wholes. Exactly how he does so is still obscure, and the mechanism probably varies widely from one operator to another. In any event transliteration is impossible if each mark and space is considered individually. MAUDE therefore uses contextual information, but far less than is available to a trained operator. The machine program knows all the standard Morse characters and a few compound ones, but no syllables or words. A trained operator, on the other hand, hears the characters themselves embedded in a meaningful context.

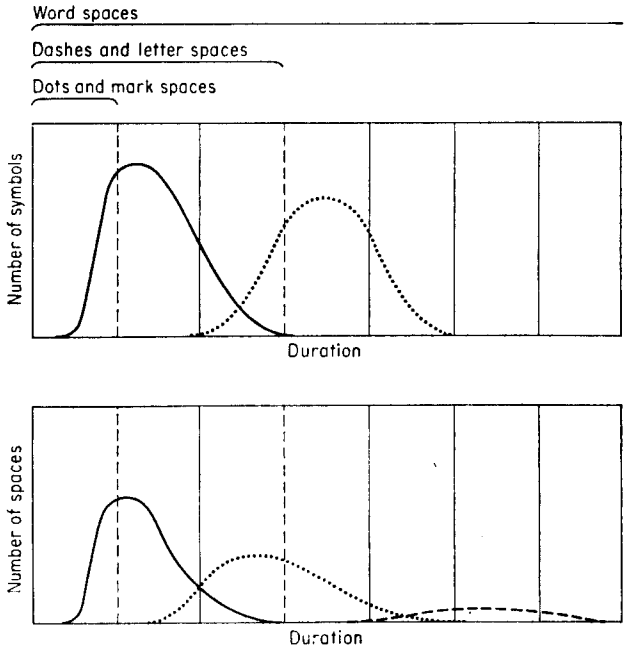


Figure 1. Variability of Morse code sent by a human operator is illustrated in these curves. Upper graph shows range of durations for dots (solid curve) and dashes (dotted curve) in a message. Lower graph gives the same information for spaces between marks within a character (solid curve), spaces between characters (dotted curve) and between words (dashed curve). Ideal durations are shown by brackets at top and vertical broken lines.

Empirically it is easier to distinguish between the two kinds of mark than among the three kinds of space. The main problem for any mechanical Morse translator is to segment the message into its characters by identifying the character spaces. MAUDE begins by assuming that the longest of each six consecutive spaces is a character space (since no Morse character is more than six marks long), and the shortest is a mark space. It is important to note that although the former rule follows logically from the structure of the ideal code, and that the latter seems quite plausible, their effectiveness can be demonstrated only by experiment. In fact the rules fail less than once in 10,000 times.

The decoding process is as follows. (See Fig. 2.) The marks and spaces, received by the machine in the form of electrical pulses, are converted into a sequence of numbers measuring their duration. (For technical reasons these numbers are then converted into their logarithms.) The sequence of durations representing spaces is processed first. The machine examines each group of six (spaces one through six, two through seven, three through eight and so on), recording in each the longest and shortest durations. When this process is complete, about 75 per cent of the character spaces and about 50 per cent of the mark spaces will have been identified.

To classify the remaining spaces a threshold is computed. It is set at the most plausible dividing line between the range of durations in which mark spaces have been found and the range of the identified character spaces. Every unclassified number larger than the threshold is then identified as a character space; every one smaller than the threshold, as a mark space.

Now, by a similar process, the numbers representing marks are identified as dots and dashes. Combining the classified marks and spaces gives a string of tentative segments, separated by character spaces. These are inspected and compared to a set of proper Morse characters stored in the machine. (There are about 50 of these, out of the total of 127 possible sequences of six or fewer marks.) Experience has shown that when one of the tentative segments is not acceptable, it is most likely that one of the supposed mark spaces within the segment should be a character space instead. The program reclassifies the longest space in the segment as a character space and examines the two new characters thus formed. The procedure continues until every segment is an acceptable character, whereupon the message is printed out.

In the course of transmitting a long message, operators usually change speed from time to time. MAUDE adapts to these changes. The computed thresholds are local, moving averages that shift with the general lengthening or shortening of marks and spaces. Thus a mark of a certain duration could be classified as a dot in one part of the message and a dash in another.

MAUDE's error rate is only slightly higher than that of a skilled human

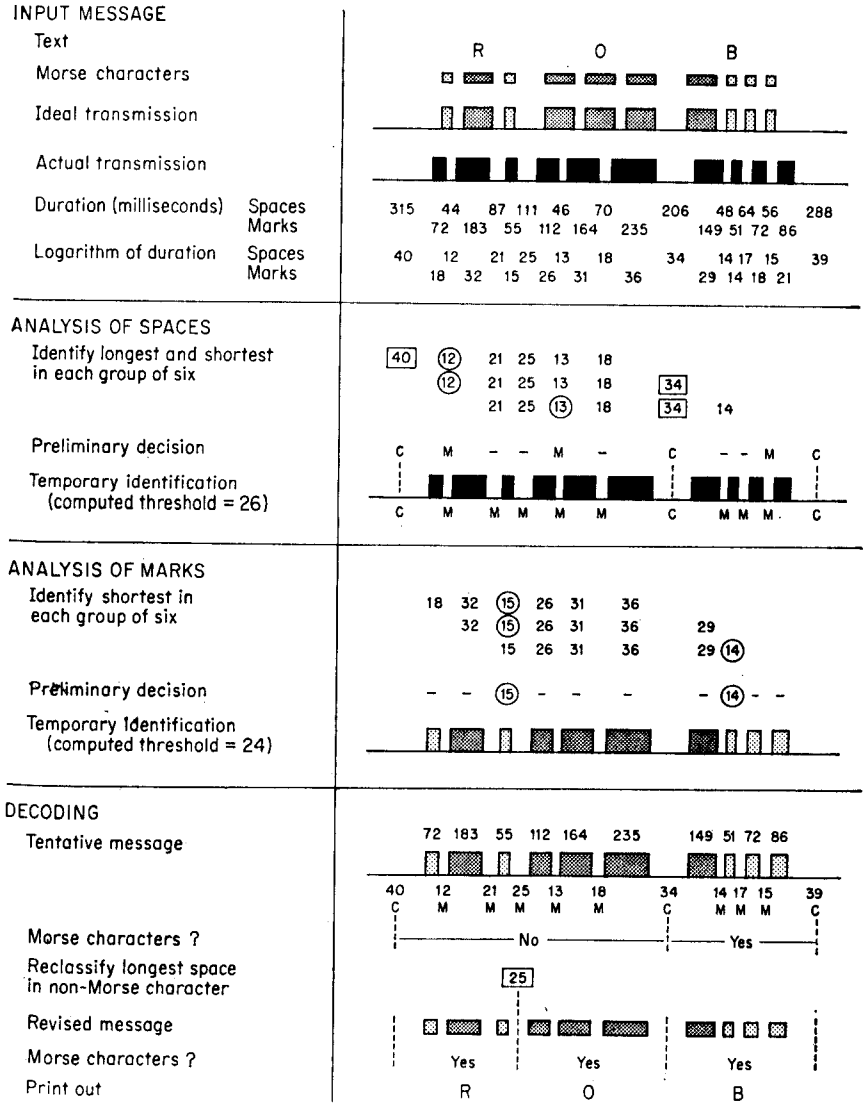


Figure 2. "MAUDE" program described in text, translates Morse code. Marks identified as dots are shown in light color; marks identified as dashes in dark color. Unidentified marks are in black. Character spaces are denoted by C; mark spaces, by M. A circle around a number indicates that it is the smallest in a group; a rectangle means it is the largest. Analysis of spaces and marks proceeds by an examination of successive groups of six throughout the message. The table shows only the first three such groups in each case.

operator. Thus it is at least possible for a machine to recognize patterns even where the basis of classification is variable and not fully specified in advance. Moreover, the program illustrates an important general point. Its success depends on the rules by which the continuous message is divided into appropriate segments. Segmentation seems likely to be a primary problem in all mechanical pattern recognition, particularly in the recognition of speech, since the natural pauses in spoken language do not generally come between words. MAUDE handles the segmentation problems in terms of context, and this will often be appropriate. In other respects MAUDE does not provide an adequate basis for generalizing about pattern recognition. The patterns of Morse code are too easy, and the processing is rather specialized.

Our second example deals with a more challenging problem: the recognition of hand-printed letters of the alphabet. The characters that people print in the ordinary course of filling out forms and questionnaires are surprisingly varied. Gaps abound where continuous lines might be expected; curves and sharp angles appear interchangeably; there is almost every imaginable distortion of slant, shape and size. Even human readers cannot always identify such characters; their error rate is about 3 per cent on randomly selected letters and numbers, seen out of context.

The first step in designing a mechanical reader is to provide it with a means of assimilating the visual data. By nature computers consider information in strings of bits: sequences of zeros and ones recorded in on-off devices. The simplest way to encode a character into such a sequence is to convert it into a sort of halftone by splitting it into a mesh or matrix of squares as fine as may be necessary. Each square is then either black or white—a binary situation that the machine is designed to handle. Making such halftones presents no problem. For example, an image of the letter could be projected on a bank of photocells, with the output of each cell controlling a binary device in the computer. In the experiments to be described here the appropriate digital information from the matrix was recorded on punch cards and was fed into the computer in this form.

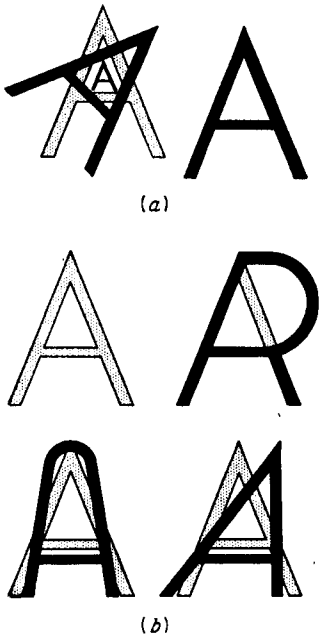
Once this sequence of bits has been put in, how shall the program proceed to identify it? Perhaps the most obvious approach is a simple matching scheme, which would evaluate the similarity of the unknown to a series of ideal templates of all the letters, previously stored in digital form in the machine. The sequence of zeros and ones representing the unknown letter would be compared to each template sequence, and the number of matching digits recorded in each case. The highest number of matches would identify the letter.

In its primitive form the scheme would clearly fail. Even if the unknown were identical to the template, slight changes in position, orientation or size could destroy the match completely. (See Fig. 3*a*.) This difficulty has

long been recognized, and in some character-recognition programs it has been met by inserting a level of information-processing ahead of the template-matching procedure. The sample is shifted, rotated and magnified or reduced in order to put it into a standard, or at least a more tractable, form.

Although obviously an improvement over raw matching, such a procedure is still inadequate. What it does is to compare shapes rather successfully. But letters are a good deal more than mere shapes. Even when a sample has been converted to standard size, position and orientation, it may match a wrong template more closely than it matches the right one. (See Fig. 3*b*.)

Nevertheless the scheme illustrates what we believe to be an important general principle. The critical change was from a program with a single level of operation to a program with two distinctly different levels. The first level shifts, and the second one matches. Such a hierarchical structure is forced on the recognition system by the nature of the entities to be recognized. The letter A is defined by the set of configurations that people call A, and their selections can be described—or imitated—only by a multilevel program.



(*a*)

(*b*)

Figure 3. (*a*) Template matching cannot succeed when the unknown letter (color) has the wrong size, orientation, or position. The program must begin by adjusting sample to standard form. (*b*) Incorrect match may result even when sample (gray) has been converted to standard form. Here R matches A template more closely than do samples of the correct letter.

We have said that letter patterns cannot be described merely as shapes. It appears that they can be specified only in terms of a preponderance of certain *features*. Thus A tends to be thinner at the top than at the bottom; it is roughly concave at the bottom; it usually has two main strokes more vertical than horizontal, one more horizontal than vertical, and so on. All these features taken together characterize A rather more closely than they characterize any other letter. Singly none of them is sufficient. For example, W is also roughly concave at the bottom, and H has a pattern of horizontal and vertical strokes similar to that described for A. Each letter has its own set of probable features, and a successful character recognizer will determine which set is the best fit to an unknown sample.

So far nothing has been said about how the features are to be determined and how the program will use them. The template-matching scheme represents one approach. Its "features," in a sense, are the individual cells of the matrix representing the unknown sample, and its procedure is to match them with corresponding cells in the template. Both features and procedures are determined by the designer. We have seen that this scheme will not succeed. In fact, any system must fail if it tries to specify every detail of a procedure for identifying patterns that are themselves defined only ostensively. A pattern-recognition system must learn. But how much?

At one extreme there have been attempts to make it learn, or generate, everything: the features, the processing, the decision procedure. The initial state of such a system is called a "random net." A large number of on-off computer elements are multiply interconnected in a random way. Each is thus fed by several others. The thresholds of the elements (the number of signals that must be received before the element fires) are then adjusted on the basis of performance. In other words, the system learns by reinforcing some pathways through the net and weakening others.

How far a random net can evolve is controversial. Probably a net can come to act as though it used templates. However, none has yet been shown capable of generating features more sophisticated than those based, like templates, on single matrix cells. Indeed, we do not believe that this is possible.

At present the only way the machine can get an adequate set of features is from a human programmer. The effectiveness of any particular set can be demonstrated only by experiment. In general there is probably safety in numbers. The designer will do well to include all the features he can think of that might plausibly be useful.

A program that does not develop its own features may nevertheless be capable of modifying some subsequent level of the decision procedure, as we shall see. First however, let us consider that procedure itself. There are two fundamentally different possibilities: sequential and parallel processing. In sequential processing the features are inspected in a predetermined

order, the outcome of each test determining the next step. Each letter is represented by a unique sequence of binary decisions. To take a simple example, a program to distinguish the letters A, H, V and Y might decide among them on the basis of the presence or absence of three features: a concavity at the top, a crossbar and a vertical line. The sequential process would ask first: "Is there a concavity at the top?" If the answer is no, the sample is A. If the answer is yes, the program asks: "Is there a crossbar?" If yes, the letter is H; if no, then: "Is there a vertical line?" If yes, the letter is Y; if no, V. (See Fig. 4.)

In parallel processing all the questions would be asked at once, and all the answers presented simultaneously to the decision maker. (See Fig. 5.) Different combinations identify the different letters. One might think of the various features as being inspected by little demons, all of whom then shout the answers in concert to a decision-making demon. From this conceit comes the name "Pandemonium" for parallel processing.

Of the two systems the sequential type is the more natural for a machine. Computer programs are sequences of instructions, in which choices or alternatives are usually introduced as "conditional transfers": Follow one set of instructions if a certain number is negative (say) and another set of instructions if it is not. Programs of this kind can be highly efficient, especially in cases where any given decision is almost certain to be right. But in "noisy" situations sequential programs require elaborate checking and backtracking procedures to compensate for erroneous decisions.

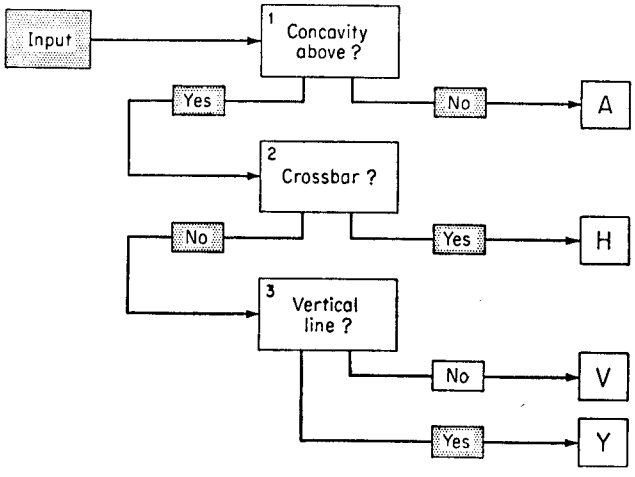


Figure 4. Sequential-processing program for distinguishing four letters, A, H, V and Y, employs three test features: presence or absence of a concavity above, a crossbar, and a vertical line. The tests are applied in order, with each outcome determining the next step.

Parallel processing, on the other hand, need make no special allowance for error and uncertainty.

Furthermore, some features are simply not subject to a reasonable dichotomy. An A very surely has a crossbar, an O very surely has not. But what about B? The most we can say is that it has more of a crossbar than O, and less than A. A Pandemonium program can handle the situation by having the demons shout more or less loudly. In other words, the information flowing through the system need not be binary; it can represent the quantitative preponderance of the various features.

Still another advantage of parallel processing lies in the possibility of making small changes in a network for experimental purposes. In typical sequential programs the only possible changes involve replacing a zero with a one, or vice versa. In parallel ones, on the other hand, the weight given to crossbarness in deciding if the unknown is actually B may be changed by as small an amount as desired. Experimental changes of this kind need not be made by the programmer alone. A program can be designed to alter internal weights as a result of experience and to profit from its mistakes. Such learning is much easier to incorporate into a Pandemonium than into a sequential system, where a change at any point has grave consequences for large parts of the system.

Parallel processing seems to be the human way of handling pattern

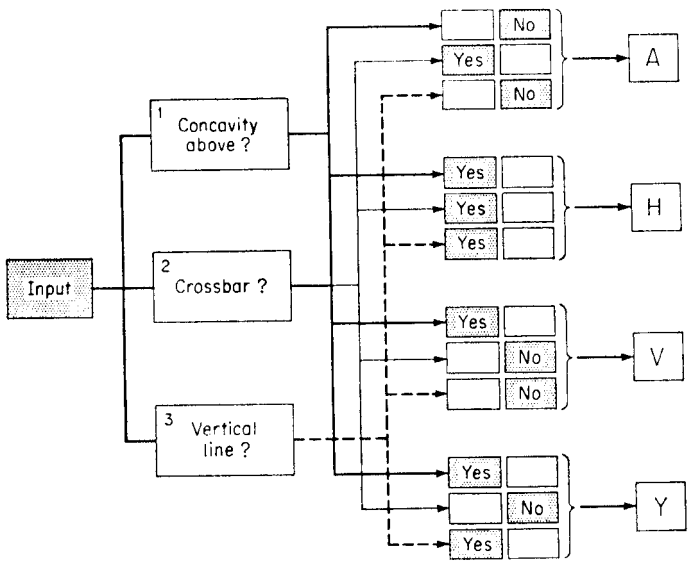


Figure 5. Parallel-processing program uses the same test features as the sequential program in Fig. 4, but applies all tests simultaneously and makes decision on the basis of the combined outcomes. The input is a sample of one of the letters A, H, V and Y.

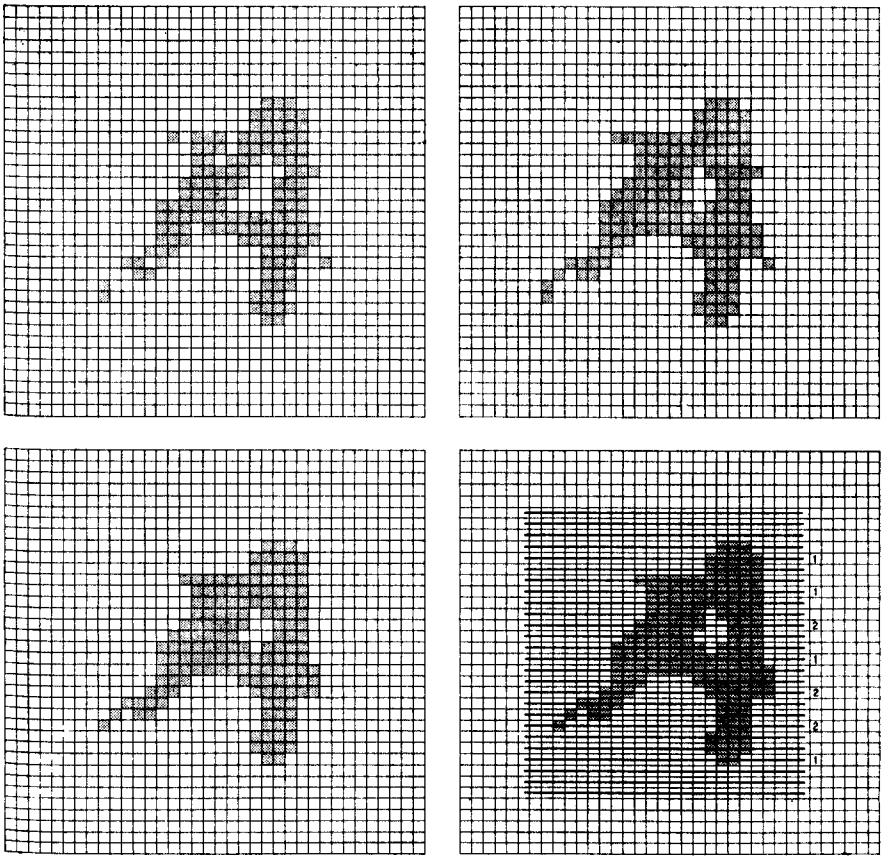


Figure 6. Hand-printed letter A is processed for recognition by a computer. Original sample is placed on grid and converted to a cellular pattern by completely filling in all squares through which lines pass (top left). The computer then cleans up the sample, filling in gaps (top right) and eliminating isolated cells (botton left). The program tests the pattern for a variety of features. The test illustrated here (bottom right) is for the maximum number of intersections of the sample with all horizontal lines across the grid.

recognition as well. Speech can be understood if all acoustic frequencies above 2000 cycles per second are eliminated, but it can also be understood if those below 2000 are eliminated instead. Depth perception is excellent if both eyes are open and the head is held still; it is also excellent if one eye is open and the head is allowed to move.

A Pandemonium system that learns from experience has been tested by Worthie Doyle of the Lincoln Laboratory. At present it is programmed to identify 10 hand-printed characters, and has been tested on samples of A, E, I, L, M, N, O, R, S and T. The program has six levels: (1) input,

Type of test and designation		Outcome	A	E	I	L	M	N	O	R	S	T
Horizontal and vertical cross-sections	HOMSXC	3	.083	.070			.250	.347	.097	.056		.097
	VEMSXC	3	.073	.339			.040		.008	.194	.258	.089
	HORUNS	2111111		.500						.500		
	VERUNS	2111111						1.000				
Strokes	HORSTR	1	.182	.006	.125	.125	.125	.146	.016	.057	.016	.203
	VERSTR	2	.178	.007			.170	.207	.229	.207		
Edge lengths and ratios	SEDGE	1	.267	.007		.014	.158	.115	.007	.165		.266
	WEDGE	1	.083	.071	.024	.024	.035	.012		.047	.318	.389
	NEDGE	2	.259	.024	.153	.024	.106	.106	.071	.059	.189	.012
	EEDGE	4	.232		.161		.214	.286	.107			
	NO:SOU	4	.513				.205	.077		.128		.077
	EA:WES	1	.055	.400		.309	.018	.036		.163		.018
Profiles	SCUCAV	3	.150				.800	.050				
	WESCAV	2	.047	.094	.023	.012	.023	.035	.035	.059	.412	.259
	NORCAV	1	.133	.177	.100	.092	.004		.133	.108	.116	.137
	EASCAV	1	.155	.005	.115	.095	.105	.130	.170	.010	.050	.165
	SOUBOT	220	.268	.106		.068	.159	.167	.008	.220	.008	
	WESBOT	221	.030	.030	.061						.364	.515
	NORBOT	121	.290	.145					.354	.042	.042	.125
	EASBOT	121	.326				.020	.102	.266	.020	.245	.020
Internal structure	SBOTSG	2	.250	.008		.016	.125	.141	.219	.203	.039	
	WBOTSG	1	.161	.076	.090	.099	.108	.121	.063	.081	.045	.157
	NBOTSG	1	.119	.190	.111	.102	.013	.018	.089	.040	.159	.159
	EBOTSG	1	.147	.058	.098	.103	.103	.121	.062	.071	.076	.061
	SOUBEN	20					.333	.167				.500
	WESBEN	10	.198	.143	.011	.022	.121	.132	.011	.099	.022	.241
	NORBEN	10	.169	.180		.135	.079			.146	.247	.045
	EASBEN	10	.211	.012	.012	.118	.176	.106		.176		.188
Total score			4.579	2.648	1.084	1.358	3.490	3.622	1.945	2.851	2.606	3.823

Figure 7. Recognition program for hand-printed letters applies the 28 feature tests listed by code name at left. Names represent such features as maximum intersection with horizontal line (HOMSXC), concavity facing south (SOUCAV), and so on. Figures in right-hand section are relative probabilities of all letters for each test outcome. The program decides on the letter with the largest total of all probabilities. In the example shown here the decision is for the letter A, with a probability total of 4.579.

(2) cleanup, (3) inspection of features, (4) comparison with learned-feature distribution, (5) computation of probabilities and (6) decision. The input is a 1024-cell matrix, 32 on a side. At the second level the sample character is smoothed by filling in isolated gaps and eliminating isolated patches. (See Fig. 6.)

Recognition is based on such features as the relative length of different edges and the maximum number of intersections of the sample with a horizontal line. (The computer "draws" the lines by inspecting every horizontal row in the matrix, and recognizes "intersections" as sequences of ones separated by sequences of zeros.) No single feature is essential to recognition, and various numbers of them have been tried. The particular program shown here uses 28. (See Fig. 7.)

Every letter fed into the machine is tested for each of the features. During the learning phase a number of samples of each of the 10 letters is presented and identified. For every feature the program compiles a table or "census." It tests each sample and enters the outcome under the appropriate letter. When the learning period is finished, the table shows how many times each outcome occurred for each of the 10 letters. Figure 8, which refers to maximum intersections with a horizontal line, represents the experience gained from a total of 330 training samples. It shows, for example, that the outcome (three intersections) occurred 72 times distributed among six A's, five E's, 18 M's, 25 N's, seven O's, four R's, seven T's and no other letters. The other possible outcomes are similarly recorded.

Next the 28 censuses are converted to tables of estimated probabilities,

Letter	Samples	Outcome			
		1	2	3	4
A	39		33	6	
E	46	6	35	5	
I	25	25			
L	24	7	17		
M	24			18	6
N	28		2	25	1
O	34		27	7	
R	33		28	4	1
S	38	8	30		
T	39	10	22	7	
Total	330	56	194	72	8

Figure 8. "CENSUS" represents information learned by letter-recognition program during training period. This table summarizes the outcomes of the test for maximum number of intersections with a horizontal line, applied to a total of 330 identified samples in the learning process.

by dividing each entry by the appropriate total. Thus the outcome—three intersections—comes from an A with a probability of .083 (6/72); an E, with a probability of .070 (5/72), and so on.

Now the system is ready to consider an unknown sample. It carries out the 28 tests and "looks up" each outcome in the corresponding feature census, entering the estimated probabilities in a table. Then the total probabilities are computed for each letter. The final decision is made by choosing the letter with the highest probability.

This program makes only about 10 per cent fewer correct identifications than human readers make—a respectable performance, to be sure. At the same time, the things it cannot do point to the difficulties that still lie ahead. We would emphasize three general problems: segmentation, hierarchical learning and feature generation.

Characters must be fed in one at a time. The program is unable to segment continuous written material. The problem will doubtless be relatively easy to solve for text consisting of separate printed characters, but will be more formidable in the case of cursive script.

The program learns on one level only. The relation between feature presence and character probability is determined by experience; everything else is fixed by the designer. It would certainly be desirable for a character recognizer to use experience for more general improvements: to change its cleanup procedures, alter the way probabilities are combined and refine its decision process. Eventually we look to recognition of words; at this point the program will have to learn a vocabulary so that it can use context in identifying dubious letters. At the moment, however, neither we nor any other designers have any experience with the interaction of several levels of learning.

The most important learning process of all is still untouched: No current program can generate test features of its own. The effectiveness of all of them is forever restricted by the ingenuity or arbitrariness of their programmers. We can barely guess how this restriction might be overcome. Until it is, "artificial intelligence" will remain tainted with artifice.