CHAPTER 29

AN OPTICALOGICAL
SELF-ORGANIZING RECOGNITION SYSTEM

George G. Lendaris and Gordon L. Stanley
GM Defense Research Laboratories
Mathematics and Evaluation Studies Department
Santa Barbara, California

1. Introduction

This paper discusses a pattern recognition system that incorporates an electro-optical preprocessor and a postprocessor which is an adaptive logic network.

As a starting point for our discussion, we state two well-known facts:

1. The Fraunhofer diffraction pattern of an aperture is the two-dimensional Fourier transform of the transmission function of that aperture.

2. Except for a constant phase factor, the Fourier transform of any function is independent of a shifting of that function.

During the last decade these facts have been used extensively in the development of spatial filtering techniques in coherent optical systems. These filtering techniques have been applied, among other things, to the recognition problem.

In this paper the point of view of sampling (for recognition), rather than filtering, is taken. Just as the servo engineer samples the frequency response characteristics of a “black box” to identify the box, we sample the frequency spectrum of a pattern to identify the pattern. The servo engineer learns to associate certain frequency responses with certain kinds of transfer functions (his characterization of a linear black box); in our case, we shall teach a network of adjustable binary logic elements (pupil) to associate given sets of frequency samples with certain binary codes (our characterization of a pattern). Having learned this, the pupil will later be able to
identify a given pattern on the basis of the frequency samples it receives. As in the case of the servo engineer, we want the pupil to identify correctly a pattern similar to, but not exactly like, the ones encountered during training.

Implicit in the word teach is the necessity of a scheme for teaching. We call such a scheme a teaching algorithm. If the network contains many interacting adjustable elements, the most difficult problem facing a training algorithm is the preservation of good information within the pupil, once the pupil obtains it. For example, when the pupil gives the wrong output for a given input, which elements are at fault? To answer this and many other questions that arise, those properties of the network that are determined by the interconnection pattern (structure) of the elements should be investigated.

We have begun such an investigation and obtained several theoretical results dealing with the class of functions which a particular structure, called a LADICAN, can realize. One of the more important results of this investigation was the development of a training algorithm for two-element LADICANs which assures attainment of a solution in a finite number of iterations, if a solution exists. This algorithm has been extended for training n-element LADICANs where the input set is unconstrained, and a limited extension has been made for the more usual case where the input set is constrained.

For reasons to be cited later, the authors used the universal logic element as the basic adjustable element for these investigations. The number of stored variables required for an m-input universal element is 2\(^m\). Therefore, as m becomes large, a single universal element becomes unfeasible, requiring in its stead a network of smaller universal elements. In addition, the more adjustable elements a network contains, the more difficult the training task. For these reasons, and for reasons of sheer economy, the number of inputs to a network of universal logic elements cannot be too large.

Although in many applications the number of inputs is large, in most of these situations not all possible configurations of these inputs are of interest—that is, the input environment is constrained—so that this large number of variables is not required for representing the input set. Hence, a preprocessor performing a many-to-few transformation may be employed. The major task of such a preprocessor in going to a smaller number of variables is to keep distinct members of the (constrained) input set distinct, or at least to keep members of distinct classes distinct. (One way to realize the many-to-few transformation of the preprocessor is to use sampling techniques.)

* Defined in Sec. III.
† To the authors’ knowledge, this is the first such training algorithm for a network of more than one interacting adjustable binary logic element.
‡ These are, respectively, algorithms 1, 2, and 3 of Reference 12.

In addition to having the preprocessor’s transformation be many-to-few, we may desire to have the transformation be invariant to certain properties of the input set. For example, in the realm of pattern recognition it could be of interest to have the transformation invariant under translation and, in some cases, invariant under rotation and/or scale change.

Whatever the operation of the preprocessor, its output must be a binary representation of the input pattern, which then serves as the input to the pupil. The pupil is trained to perform a further many-to-few transformation, whereupon each member of an input class is assigned the same output code, and each distinct class is given a distinct code (Fig. 1).

**Fig. 1. Recognition system.**

The primary reason for following the preprocessor with a (teachable) pupil is the pupil’s potential ability to generalize well over the patterns it does not “see” during training. In any realistic application, not all the possible pertinent patterns will be available during the design phase (or the training phase, in our case) of a recognition system. Thus it is important to have a device that is capable of giving a correct response for patterns that are not used in the design (training) phase.

In the sequel, an application of these techniques to a character recognition task is described: the preprocessor in Sec. II, the pupil in Sec. III, and experimental results in Sec. IV.

The term “opticalical” in the title refers to the optical nature of the preprocessor and the logic elements of the pupil.

II. Preprocessor

We assume that the pattern given for identification is in the form of a photographic transparency, and speak of the pattern as the amplitude transmission function of the transparency. Facts (1) and (2) of Sec. I tell us that the Fraunhofer diffraction pattern (FDP) of the given pattern will be independent of translation of that pattern (within the optical system’s aperture). Therefore, if the FDP is sampled appropriately, a many-to-few transformation insensitive to translation can be performed.

An optical configuration for generating the FDP of an input pattern is shown in Fig. 2. As usual, the FDP is centered on the optical axis of the

* See Sec. III.
† A more complete treatment of the pupil and the training algorithm is given in Reference 12.
system. Since the Fourier transform preserves rotation, if the input pattern is rotated the FDP is rotated about the optical axis but remains otherwise unchanged.

In contrast to the electronic-communication-system analog, the optical image in the frequency plane has two degrees of freedom—that is, two dimensions instead of one. Because of this, the sampling we perform in the frequency plane can be done in a greater variety of ways than in the communication system.

As an example, we can determine the contributions of a particular frequency component, independent of direction, in the input pattern. This can be accomplished in the optical frequency plane by integrating the light amplitude along a circle of appropriate radius centered on the optical axis, and may be repeated for other frequency components by choosing appropriate radii. The resulting data will be independent of rotation.

In principle, such a sampling scheme is carried out by integrating over a (mathematical) line. In practice, of course, this cannot be done. But, because we want only a small number of samples and we want these samples to be insensitive to minute variations (e.g., for noise considerations), it is desirable to make the lines relatively wide to include a band of frequencies simultaneously. In this way, a certain amount of averaging is effected. If, say, \( m \) samples are desired, each line can be made \( 1/m \) units wide, where \( 1 \) is the normalized size of the interesting portion of the FDP. We call these lines sampling windows. Typical windows are shown in Fig. 3.

Currently, the integration mentioned above is performed optically and the result is converted to an electrical voltage. After these analog voltages representing the samples are obtained, the problem remains of digitally encoding them with a small number of bits—say one, two, or three bits. For the one-bit encoding, one threshold value has to be determined; for the two-bit encoding, three threshold values; and for the three-bit encoding, seven threshold values. There are many potential bases upon which such threshold determinations can be made—selection will depend upon many factors, including the application at hand.

We direct our attention to the application where the patterns to be recognized are the letters of the English alphabet. We generate and sample the FDP of each of the 26 letters of one type font, yielding \( 26m \) pieces of analog data, where \( m \) is the number of sampling windows. We construct a 26-by-\( m \) matrix of these data, where the rows correspond to the letters, and the columns to the sampling windows.

One possibility for determining the encoding thresholds is to scan a row of this matrix to find the maximum and minimum values and establish thresholds to separate the resulting range into two, four, or eight equal parts. Then we assign the appropriate one-, two-, or three-bit code to each entry in the given row, repeating for all rows. Alternatively, the thresholds can be made to separate the members of the row into two, four, or eight equal population classes (e.g., median value for the one-bit case), again repeating for each row. Another method is to perform the above operations on the columns instead of the rows of the matrix. Or, these operations can be performed on the entire matrix, rather than on just rows or columns. The authors have experimentally investigated all six methods. Some of the results are cited in Sec. IV.

As an aid to understanding the over-all operation of the preprocessor, an annotated schematic diagram of the system used initially to determine the feasibility of this approach is shown in Fig. 4.
III. Pupil

As mentioned in Sec. I, we have begun a program of investigating structure-dependent properties of networks of adaptive logic elements. So that the properties thus obtained would reflect constraints imposed only by the structure and not by the elements, universal logic elements were used as the basic element in these investigations. A universal logic element is an element with $m$ binary inputs and one binary output capable of performing all $2^m$ possible switching functions on its $m$ binary inputs. Which of the $2^m$ functions the element performs is determined by internal parameter values (changed during training). An $m$-input universal element is designated by $U_m$.

The theoretical tool used in these investigations was the Ashenhurst-Curtis theory of decomposition of switching functions (hereafter called AC theory). As a starting point, we decided to investigate a cascade type of network (because of the relative ease of applying the AC theory) and to begin with a disjunctive partitioning on the inputs (each external input goes to only one element within the net). This latter decision was based upon a conjecture put forth by Curtis that if a function can be realized disjunctively, then this realization is probably minimal.

The network investigated is shown in Fig. 5. As indicated, the number of inputs to the $j$th element is given by $i_j$. A switching function realizable by such a network is called disjunctively decomposable—we therefore call this network a labeled disjunctive cascade network, where the word labeled means that a definite assignment is made for each external input. For notational convenience, we use the acronym LADICAN in place of labeled disjunctive cascade network.

![Fig. 5. Pupil: labeled disjunctive cascade network of $n$ universal elements.](image)

The structure-dependent properties thus far determined for LADICANs are reported in References 11 and 12. Of interest here is the algorithm developed for training LADICANs. This algorithm assures that a solution will be reached in a finite number of iterations, if a solution exists. Basically, the training of the pupil proceeds as follows: An input is given to the pupil, and the resulting output is compared with what the output should be; if it is correct, a positive reinforcement is applied (to all the elements within the net); if it is incorrect, a negative reinforcement is applied. The present theorem of convergence applies to the punish-only type of algorithm—a negative reinforcement is applied when the pupil gives an incorrect response to an input, but none when the response is correct; however, a reward-punish algorithm is being successfully used experimentally.

The LADICAN is used as the pupil in the present character recognition system. The primary reason for using such a device in a recognition system is its potential ability to generalize well. In Reference 12, we discuss the notion of generalization at some length and define a means of expressing generalization quantitatively. We will describe this notion briefly.

Referring to Fig. 6, let $A$ be the set of all possible patterns of $n$ binary variables. Let $C$ be the collection of the patterns that we “care” about—

![Fig. 6. Model for generalization discussion.](image)

this discussion, we agree to “care” about a pattern if it (somehow) represents a typed letter of the English alphabet (e.g., output of the preprocessor). Let $C_T$ be the collection of care-terms to be used for training the pupil.

At this point, we need the following definitions. Given a function $F(x_1, x_2, \ldots, x_n)$, if a value is assigned by $F$ for each possible $n$-tuple $(x_1, x_2, \ldots, x_n)$, then $F$ is said to be a total function. If there are some $n$-tuples for which $F$ is undefined, $F$ is called a partial function. Given two functions, $F$ and $G$, with the same independent variables, if for every $n$-tuple for which $F$ is defined $G$ is also defined and assigns the same value as $F$, then $G$ is an extension of $F$, denoted $G \supseteq F$. Alternatively, $F$ is a restriction of $G$. 


Let \( D \) be a partial function which assigns to each member of \( C \) an output indicating which letter of the alphabet it is; this function is known a priori by the teacher. Let \( T \) be a partial function, defined for only the members of \( C_T \), which, like \( D \), assigns to each of these (training) patterns an output indicating which letter it is.

Let the teacher train the pupil until the pupil issues a correct output for, say, each of the patterns in set \( C_T \). Clearly, the pupil will issue some output for each input it receives, even for patterns not in \( C_T \). Thus, a function, call it \( P \), defining the input-output operation of the pupil is total. Since the pupil issues the correct output (as specified by \( T \)) for each pattern in \( C_T \), \( P \) is an extension of \( T \).

We now wish to compare the assignments of \( P \) with those of \( D \) for each of the care terms not in \( C_T \) (call this set \( C/C_T \)). Although the word “generalize” has been used to describe the case when all (or some) of these assignments are alike, this use of the word is not precise. The dictionary definition of “generalization” contains no assertion of “correctness.” Therefore, any extension of a function is a generalization of that function. The problem is to compare this generalization with some other function, partial or total, to determine the degree of concurrence, or “correctness,” over the set of care terms. No matter what the application, some standard must be used as the basis of an assertion of “correctness.”

Assuming partial function \( D \) as our standard, we compare \( P \) and \( D \) for the terms in \( C/C_T \). In practice, \( C \) will usually be a very large set. Thus, in general we will not be able to compare \( P \) and \( D \) for all the care-terms not in \( C_T \). Therefore, any judgment of how well \( P \) and \( D \) compare in \( C/C_T \) will have to be made on the basis of an examination of some subset of these patterns. For this examination, we chose a subset \( C_E \) of \( C/C_T \).

Any reasonably precise statement of generalization should include information regarding the size of \( C_T \), the size of \( C_E \), and the number of patterns in each for which \( P \) assigns the correct output. (It may turn out that even after training, the pupil will not assign the correct output for each term in \( C_T \).)

Let the number of terms in set \( C_T \) be \( t \), and in \( C_E \) be \( e \). Let \( t_e \) be the number of patterns in \( C_T \) for which the pupil gives correct answers, and let \( e_0 \) be the number of patterns in \( C_E \) for which the pupil gives correct answers. The total number of patterns presented to the pupil, then, is \( t + e \), and the proportion used during training is \( t/(t + e) \).

On this basis, we construct the following expression to express generalization quantitatively:

\[
\frac{e}{e_0} \text{ generalization via } \frac{t}{t + e} \text{ training performance on } \frac{t}{t + e} \text{ exposure}
\]

It is important to remember that this statement depends upon the exam set \( C_E \). An a priori judgment has to be made that the set \( C_E \) is representative of the set \( C/C_T \).

For comparing the generalization performance of one teacher-pupil combination on a training and exam set with that of another combination on the same sets, or of one teacher-pupil combination on several training and exam sets for a constant \( t/(t + e) \), we define a Generalization Ratio

\[
\text{GR} = \frac{e_0}{e} \frac{t}{t}
\]

When two Generalization Ratios are compared, it is assumed that either the sets \( C_T \) and \( C_E \) are held constant with variations occurring in the pupil and/or teacher, or that the pupil-teacher combination is fixed, with variations occurring in the selection of sets \( C_T \) and/or \( C_E \) for constant \( t/(t + e) \). The first assumption is made for the generalization results cited in Sec. IV.

As an example, suppose pupil one has 0.8 generalization via 0.8 training performance on a given exposure, whereas pupil two has 0.8 generalization via 1.0 training performance on the same training exposure. Which pupil did “better” at generalizing? We assert that pupil one did, because it performed as well on the patterns to which it had never been exposed as it did on those of the training set. Pupil two, however, performed less well on the exam set than on the training set. For this illustration,

\[
\text{GR (pupil one)} = 1.0 \\
\text{GR (pupil two)} = 0.8
\]

An examination of these ratios indicates that pupil one did better than pupil two at generalizing.

IV. Experimental Results

We cite a few experimental results to demonstrate the operation of this system.

A. Preprocessor

The first step in the preprocessor is the generation of the Fraunhofer diffraction patterns. The FDP’s for the letters shown in Fig. 7 are shown in

Fig. 7. Type font used for FDP’s of

\[\begin{array}{cccccccccccccccccccc}
A & B & C & D & E & F & G & H & I \\
J & K & L & M & N & O & P & Q & R \\
S & T & U & V & W & X & Y & Z
\end{array}\]

Fig. 8. An encoding of these letters via the sampling windows of Fig. 9 is shown in Fig. 10. (There are five windows, each with a 3-bit encoding, yielding a 15-bit unique code for each of the 26 letters. Note that bits one

* The teacher, here, is the training algorithm.
and four, from the left, can be removed, and the resulting 13-bit codes will still be unique.)

A considerable amount of data has been taken on five different alphabet fonts. Eleven different sets of sampling windows were used, and all six methods of encoding were applied. The resulting codes ranged from 15 bits (Fig. 10) to 96 bits. As a rule, the longer codes had a wider Hamming separation.*

![Fig. 9. Sampling windows used for encoding FDPs of Fig. 8.](image)

<table>
<thead>
<tr>
<th>WINDOW</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>O</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Fig. 10. A 15-bit encoding of the letters of Fig. 7 (and Fig. 8) via the sampling windows of Fig. 9.](image)

Several of the sampling-window/encoding combinations yielded a distinct code for each letter of a font; and when the five fonts were compared, no different letters had been given the same code. In the experimental work

* The Hamming distance between two codes is the number of bit position at which the codes differ.
to date, a different set of thresholds is computed for each font. In relation to the over-all system, this means that the preprocessor has to be "told" which font it is processing. With improvements that are to be made in the sampling process, and by processing all font data together, this problem should be overcome in future work.

B. Pupil

Various teacher-LADICAN combinations have been subjected to extensive experimentation with unconstrained environments, with the purpose of investigating the effects of various parameters on the relative efficiency of attaining a solution. (These results are given in Reference 11.)

Some experiments were performed to investigate the LADICAN's ability to generalize well. The authors have conjectured that, for a given number of inputs, the longer the LADICAN used, the larger the generalization ratio will be.

Data for only five fonts were generated via the preprocessor. To give the LADICAN a larger sample over which to generalize, a different method of generating data was used for these experiments. A $4 \times 4$ grid (Fig. 11) was used, and 41 different block C's and 41 different block T's were drawn on the grid in a black-on-white fashion. An encoding of these letters was effected by assigning a 1 to a dark square, and a 0 to a light square. The resulting 16-bit codes were used as inputs to the LADICAN in the generalization experiments.

![Fig. 11. The $4 \times 4$ grid used for generating generalization data.](image)

Fig. 12. LADICANs used in generalization experiment.

A training set of 58 patterns was arbitrarily chosen from the 82 patterns; the remaining 24 patterns were used for the exam set. Three networks were

* For reasons to be published later.

![Fig. 12. LADICANs used in generalization experiment.](image)
four alphabet fonts. Unfortunately, in this data set the coding for the letter Y of font 1 (denoted Y₁) turned out to be the same as S₁, and the coding for S₂ the same as F₁. These results are tabulated in Fig. 15. We list the tₜ/t performance on the basis that the entire output codes are correct (i.e., all five LADICANs are correct simultaneously), and on the basis that the individual bits are correct (each LADICAN independently correct or not). An asterisk next to the run means that two distinct letters have the same input code, and hence the pupil cannot be “blamed” for making mistakes on these letters.

<table>
<thead>
<tr>
<th>Run</th>
<th>Font</th>
<th>5 bits simultaneously</th>
<th>Individual bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>5*</td>
<td>1.2</td>
<td>0.982</td>
<td>0.989</td>
</tr>
<tr>
<td>6</td>
<td>1.3</td>
<td>0.982</td>
<td>0.997</td>
</tr>
<tr>
<td>7</td>
<td>1.4</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>8</td>
<td>2.3</td>
<td>0.982</td>
<td>0.997</td>
</tr>
<tr>
<td>9</td>
<td>2.4</td>
<td>0.982</td>
<td>0.997</td>
</tr>
<tr>
<td>10*</td>
<td>3.4</td>
<td>0.936</td>
<td>0.983</td>
</tr>
<tr>
<td>11*</td>
<td>1.2,3</td>
<td>0.962</td>
<td>0.988</td>
</tr>
<tr>
<td>12*</td>
<td>1.2,4</td>
<td>0.936</td>
<td>0.988</td>
</tr>
<tr>
<td>13*</td>
<td>1.3,4</td>
<td>0.936</td>
<td>0.988</td>
</tr>
<tr>
<td>14*</td>
<td>2.3,4</td>
<td>0.936</td>
<td>0.985</td>
</tr>
<tr>
<td>15*</td>
<td>1.2,3,4</td>
<td>0.856</td>
<td>0.967</td>
</tr>
</tbody>
</table>

Fig. 15. Results of training system to recognize fonts of English alphabet.

The importance of the data in the last column of this table is that, since so few bits are wrong, perhaps some error-correcting scheme can be devised to improve the recognition. The attendant requirement of longer codes can be realized by adding a few more LADICANs in parallel. This process might also prove fruitful for generalization.

V. Conclusions

This paper has discussed one approach to the pattern recognition problem. Basically, the approach involves the use of an electro-optical preprocessor, and a postprocessor made up of binary logic elements. At this level of description, the approach is not new. Investigators at Stanford Research Institute are working on a recognition machine which is similarly structured. It, too, incorporates an electro-optical preprocessor and an adaptive postprocessor. The differences lie in the realization of the two parts. Their preprocessor samples the original patterns, whereas ours samples the two-dimensional Fourier transform of the patterns. Their main advantage is that the pupil learns the font perfectly. However, if we consider this ambiguity as noise in the teacher-to-pupil channel, Fig. 15 shows that the pupil’s performance did not deteriorate beyond the ambiguity induced by this noise.

The processor uses the threshold logic unit as the basic binary logic element; ours uses the universal logic element. The training of their main processor depends upon a basic one-element convergence theorem; ours depends on our two-element convergence theorem (extendable to n elements under suitable conditions).

Experimental results with our system, although meager, are encouraging. Present indications are that the type of sampling windows to use depend upon the given task and, perhaps, upon the characteristics of the data on which this task is to be performed. Further, attention will have to be given to the following questions:

1. In what order are the preprocessor outputs to be connected to the postprocessor?
2. How can the postprocessor output coding be selected to enhance the chances of obtaining a function realizable by the adaptive net and/or to enhance the generalization of the postprocessor?

REFERENCES
