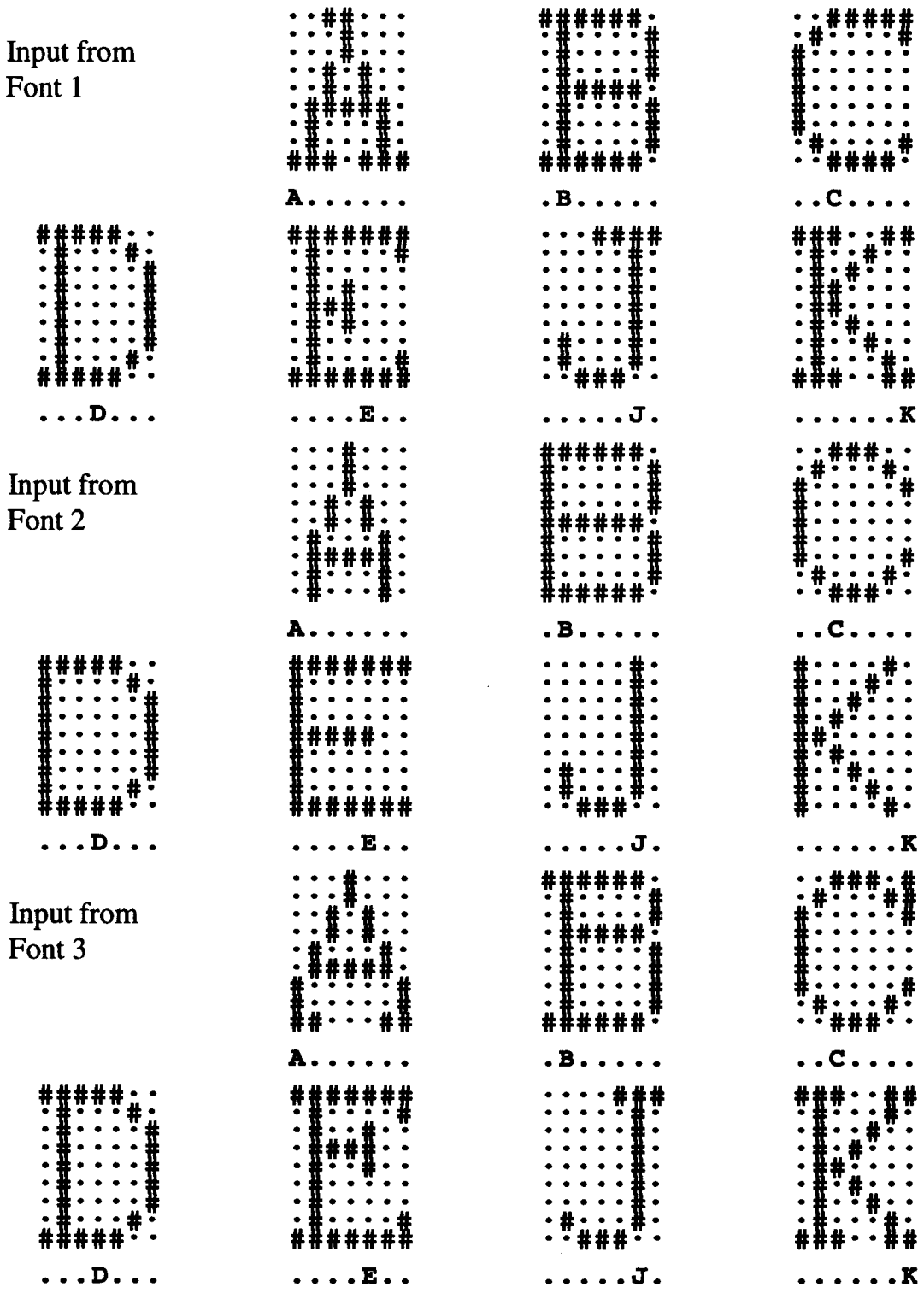

Figure 2.20 Training input and target output patterns.

the class A. In that case, the target value for each pattern is either 1 or -1; only the first component of the target vector shown is applicable. The net is as shown in Figure 2.14, and $n = 63$. There are three examples of A and 18 examples of not-A in Figure 2.20.

We could, of course, use the same vectors as examples of B or not-B and train the net in a similar manner. Note, however, that because we are using a single-layer net, the weights for the output unit signifying A do not have any interaction with the weights for the output unit signifying B. Therefore, we can solve these two problems at the same time, by allowing a column of weights for each output unit. Our net would have 63 input units and 2 output units. The first output unit would correspond to "A or not-A", the second unit to "B or not-B." Continuing this idea, we can identify 7 output units, one for each of the 7 categories into which we wish to classify our input.

Ideally, when an unknown character is presented to the net, the net's output consists of a single "yes" and six "nos." In practice, that may not happen, but the net may produce several guesses that can be resolved by other methods, such as considering the strengths of the activations of the various output units prior to setting the threshold or examining the context in which the ill-classified character occurs.

Example 2.15 A Perceptron to classify letters from different fonts: several output classes

The perceptron shown in Figure 2.14 can be extended easily to the case where the input vectors belong to one (or more) of several categories. In this type of application, there is an output unit representing each of the categories to which the input vectors may belong. The architecture of such a net is shown in Figure 2.21.

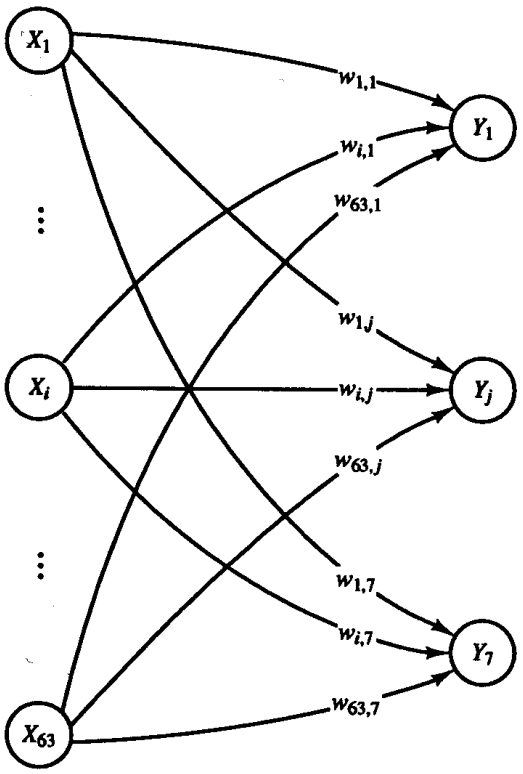

Figure 2.21 Perceptron to classify input into seven categories.

For this example, each input vector is a 63-tuple representing a letter expressed as a pattern on a 7 × 9 grid of pixels. The training patterns are illustrated in Figure 2.20. There are seven categories to which each input vector may belong, so there are seven components to the output vector, each representing a letter: A, B, C, D, E, K, or J. For ease of reading, we show the target output pattern indicating that the input was an "A" as (A · · · · · · ·), a "B" (· B · · · · · ·), etc.

The training input patterns and target responses must be converted to an appropriate form for the neural net to process. A bipolar representation has better computational characteristics than does a binary representation. The input patterns may be converted to bipolar vectors as described in Example 2.8; the target output pattern (A · · · · · · ·) becomes the bipolar vector (1, −1, −1, −1, −1, −1, −1) and the target pattern (· B · · · · · ·) is represented by the bipolar vector (−1, 1, −1, −1, −1, −1, −1).

A modified training algorithm for several output categories (threshold = 0, learning rate = 1, bipolar training pairs) is as follows:

Step 0. Initialize weights and biases
(0 or small random values).

Step 1. While stopping condition is false, do Steps 1–6.

Step 2. For each bipolar training pair **s : t**, do Steps 3–5.

Step 3. Set activation of each input unit, $i = 1, \ldots, n$:

$$x_i = s_i.$$

Step 4. Compute activation of each output unit, $j = 1, \ldots, m$:

$$y\_in_j = b_j + \sum_i x_i w_{ij}.$$

$$y_j = \begin{cases} 1 & \text{if } y\_in_j > \theta \\ 0 & \text{if } -\theta \le y\_in_j \le \theta \\ -1 & \text{if } y\_in_j < -\theta \end{cases}$$

Step 5. Update biases and weights, $j = 1, \ldots, m$; $i = 1, \ldots, n$:

If $t_j \neq y_j$, then

$$b_j(\text{new}) = b_j(\text{old}) + t_j;$$

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + t_j x_i.$$

Else, biases and weights remain unchanged.

Step 6. Test for stopping condition:
If no weight changes occurred in Step 2, stop; otherwise, continue.

After training, the net correctly classifies each of the training vectors.

The performance of the net shown in Figure 2.21 in classifying input vectors that are similar to the training vectors is shown in Figure 2.22. Each of the input

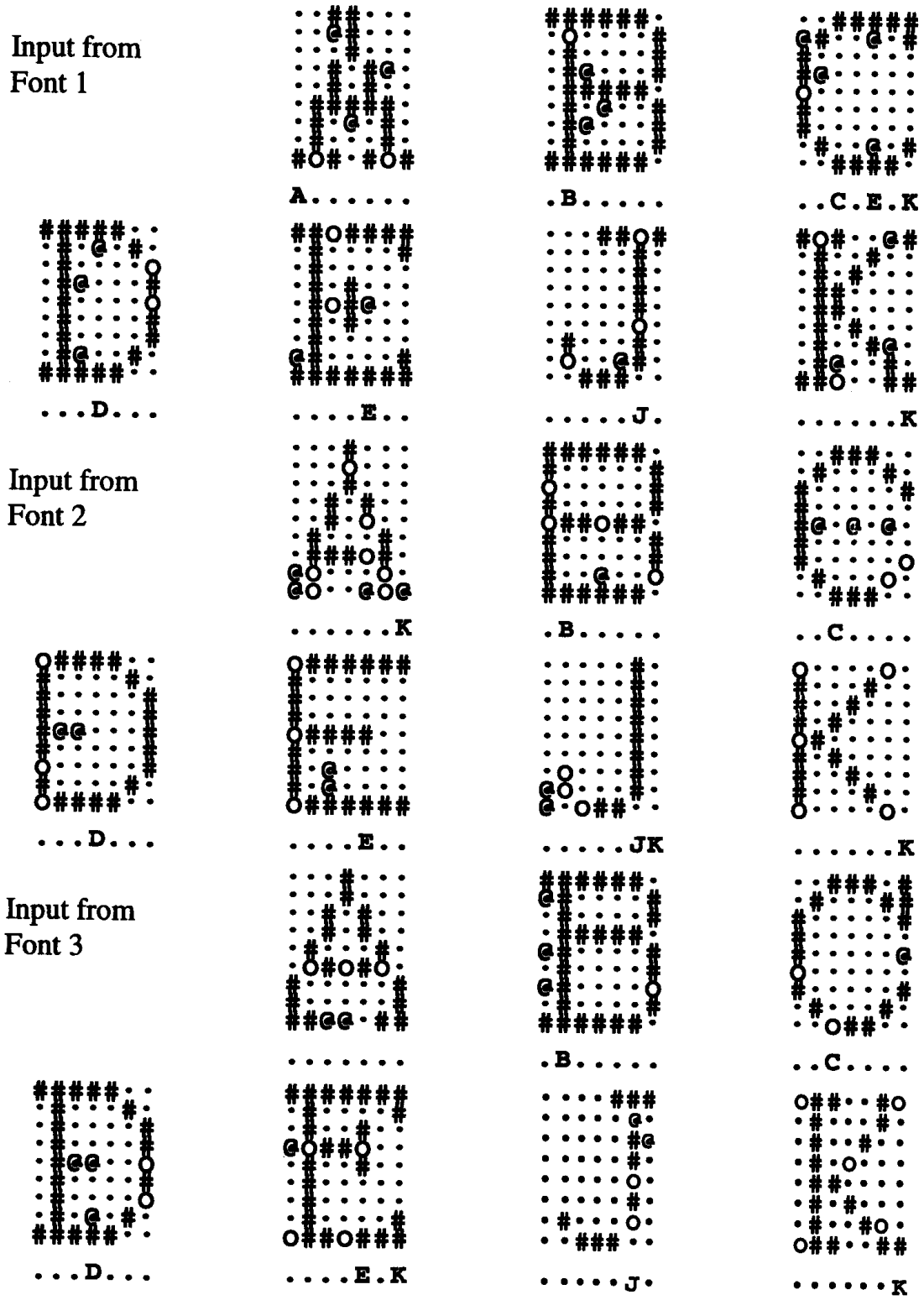

Figure 2.22 Classification of noisy input patterns using a perceptron.