PATTERN RECOGNITION

Pattern recognition (PR) concerns the description or classification (recognition) of measurements. PR is an important, useful, and rapidly developing technology which is not one technique, but rather a broad body of often loosely related knowledge and techniques. PR may be characterized as an information reduction, information mapping, or information labeling process. Historically, the two major approaches to pattern recognition are statistical (or decision theoretic), hereafter denoted StatPR, and syntactic (or structural), hereafter denoted SyntPR. Recently, the emerging technology of neural networks has provided another alternative, neural pattern recognition, hereafter denoted NeurPR. NeurPR is especially well suited for "black box" implementation of PR algorithms. Since no single technology is always the optimum solution for a given PR problem, all three are often considered in the quest for a solution.

The structure of a generic pattern recognition system is shown in Fig. 1. Note that it consists of a sensor or set of sensors, a feature extraction mechanism (algorithm), and a classification or description algorithm (depending upon the approach). In addition, usually some data which has already been classified or described is assumed available in order to train the system (the so-called training set).

PATTERNS AND FEATURES

Pattern recognition, naturally, is based upon patterns. A pattern can be as basic as a set of measurements or observations, perhaps represented in vector notation. *Features* are any extracted measurement used. Examples of low-level features are signal intensities. Features may be symbolic, numeric, or both. An example of a symbolic feature is color; an example of a numerical feature is weight (measured in pounds). Features may also result from applying a feature extraction algorithm or operator to the input data. Additionally, features may be higher-level entities, for example, geometric descriptors of either an image region or a 3-D object appearing in the image. For example, in image analysis applications (1), aspect ratio and Euler number are higher level geometric features extracted from image regions. Significant computational effort may be required in feature extraction and the extracted features may contain errors or noise. Features may be represented by continuous, discrete, or discrete-binary variables

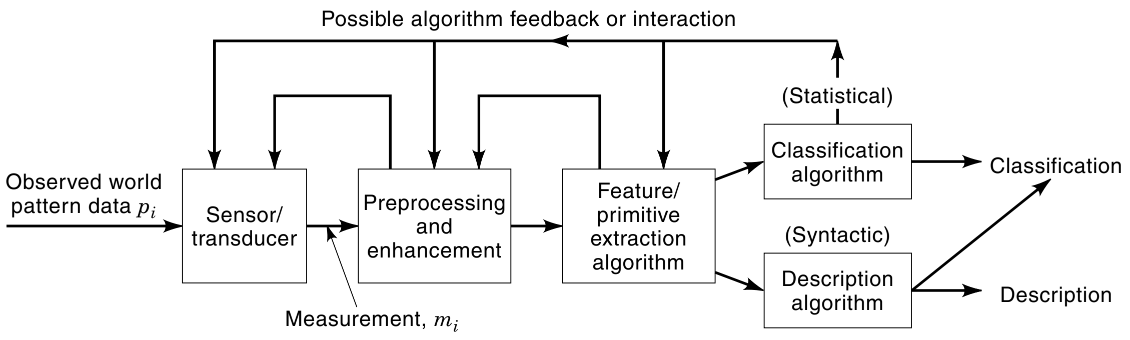
Possible algorithm feedback or interaction
(Statistical)
Observed world pattern data p_i
Sensor/ transducer
Preprocessing and enhancement
Feature/ primitive extraction algorithm
Classification algorithm
Classification
(Syntactic)
Description algorithm
Description
Measurement, m_i

Figure 1. Generic pattern recognition system elements [from (2) Copyright 1992. Reprinted by permission of John Wiley & Sons.]

Binary features may be used to represent the presence or absence of a particular attribute. The interrelated problems of feature selection and feature extraction must be addressed at the outset of any PR system design.

Statistical pattern recognition is explored in depth in numerous books. Good references are (2–10).

The Feature Vector and Feature Space

Feature vectors are typically used in StatPR and NeurPR. It is often useful to develop a geometrical viewpoint of features in these cases. Features are arranged in a d-dimensional feature vector, denoted $\boldsymbol{x}$, which yields a multidimensional feature space. If each feature is an unconstrained real number, the feature space is R^d. In other cases, for example those involving artificial neural networks, it is convenient to restrict feature space to a subspace of R^d. Specifically, if individual neuron outputs and network inputs are restricted to the range [0,1], for a d-dimensional feature vector the feature space is a unit volume hypercube in R^d.

Classification of feature vectors may be accomplished by partitioning feature space into regions for each class. Large feature vector dimensionality often occurs unless the data are preprocessed. For example, in image processing applications, it is impractical to directly use all the pixel intensities in an image as a feature vector since a 512×512 pixel image yields a $262{,}144 \times 1$ feature vector.

Feature vectors are somewhat inadequate or at least cumbersome when it is necessary to represent relations between pattern components. Often, classification, recognition, or description of a pattern is desired which is invariant to some (known) pattern changes or deviation from the "ideal" case. These deviations may be due to a variety of causes, including noise. In many cases a set of patterns from the same class may exhibit wide variations from a single exemplar of the class. For example, humans are able to recognize (that is, classify) printed or handwritten characters with widely varying font sizes and orientations. Although the exact mechanism which facilitates this capability is unknown, it appears that the matching strongly involves structural analysis of each character.

Feature Vector Overlap. Since feature vectors obtained from exemplars of two different classes may overlap in feature space, classification errors occur. An example of this is shown in Fig. 2.

Example of Feature Extraction. Consider the design of a system to identify two types of machine parts. One part, which is denoted a shim, is typically dark and has no surface intensity variation or texture. Another part, denoted a machine bolt, is predominantly bright, and has considerable surface intensity variation. For illustration, only texture and brightness are used as features, thus yielding a 2-D feature space and feature vector. We also assume these features are extracted from suitable measurements. Other possible features, such as shape, weight, and so on, may be used. The problem, as formulated, is challenging since these features are only typical of each part type. There exist cases of shims which are bright and textured and bolts which are dark and have little texture, although they are atypical, that is, they do not occur often. More important, when features overlap, perfect classification is not possible. Therefore, classification error, characterized via the probability P(error), indicates the likelihood of an incorrect classification or decision. In this example, element x_i, $i = 1, 2$, is a feature, where x_1 is measured or computed brightness and x_2 is measured or computed texture. Furthermore, w_i is a *class,* or a state of nature, where w_1 is taken to

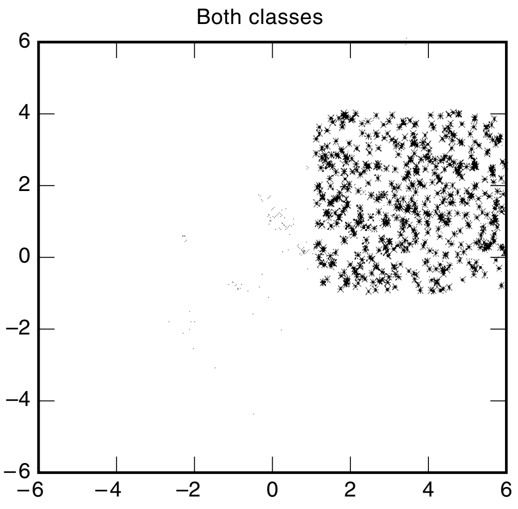

Figure 2. Example of feature vector overlap, leading to classification error.

be shim and w_2 is bolt. Feature vector overlap may occur in this example. If the underlying class is w_1 (shims), we expect typical measurements of x_1 and x_2 (brightness and texture respectively) to be small, whereas if the object under observation is from class w_2 (bolts) we expect the values of x_1 and x_2 to be, on the average, large (or at least larger than those of w_1). Of particular importance is the region where values of the features overlap. In this area errors in classification are likely. A more general cost or risk measure may be associated with a classification strategy.

Pattern Classification

Classification is the assignment of input data into one or more of c prespecified classes based upon extraction of significant features or attributes and the processing and/or analysis of these attributes. It is common to resort to probabilistic or grammatical models in classification.

Recognition is the ability to classify. Often we formulate PR problems with a (c + 1)st class, corresponding to the unclassifiable, do not know, or cannot decide class.

Description is an alternative to classification where a structural description of the input pattern is desired. It is common to resort to linguistic or structural models in description. A *pattern class* is a set of patterns (hopefully sharing some common attributes) known to originate from the same source. The key in many PR applications is to identify suitable attributes (e.g., features) and form a good measure of similarity and an associated matching process.

Preprocessing is the filtering or transforming of the raw input data to aid computational feasibility and feature extraction and minimize noise.

Noise is a concept originating in communications theory. In PR, the concept is generalized to represent a number of nonideal circumstances.

Pattern Matching

Much of StatPR, SyntPR, and NeurPR is based upon the concept of pattern similarity. For example, if a pattern, x, is very similar to other patterns known to belong to class w_1, we would intuitively tend to classify x as belonging in w_1. Quantifying similarity by developing suitable similarity measures, is often quite difficult. Universally applicable similarity measures which enable good classification are both desirable and elusive.

Measures of similarity (or dissimilarity) using feature vectors are commonly used. Distance is one measure of vector similarity. The Euclidean distance between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by

$$d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\| = \sqrt{(\boldsymbol{x}-\boldsymbol{y})^T(\boldsymbol{x}-\boldsymbol{y})}$$
$$= +\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

A related and more general metric is

$$d_p(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{d}|x_i - y_i|^p\right)^{1/p}$$

Commonly, weighted distance measures are used. An example is

$$d_w^2(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T R(\boldsymbol{x}-\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|_R^2$$

This implements upon a weighted inner product or weighted R-norm. The matrix R is often required to be positive definite and symmetric. When $\boldsymbol{x}$ and $\boldsymbol{y}$ are binary, measures such as the Hamming distance are useful.

Training

A set of typical patterns, where typical attributes or the class or structure of each is known, forms a database. This database is called the training set and denoted H. In a general sense, the training set provides significant information on how to associate input data with output decisions (i.e., classifications or structural descriptions). Training is often associated (or loosely equated) with learning. The training set is used to enable the system learn relevant information, such as statistical parameters, natural groupings, key features, or underlying structure. In SyntPR, training samples are used to learn or infer grammars.

Supervised and Unsupervised Classification

Training uses representative (and usually labeled) samples of types of patterns to be encountered in the actual application. The training set is denoted H or H_i, where the subscript denotes a training set for a specific pattern class. In some cases, the training set for class w_i contains examples of patterns in w_i (positive exemplars) as well as examples of patterns not in w_i (negative exemplars).

In this context, supervised learning or training assumes a labeled (with respect to pattern class) set, whereas in unsupervised learning the elements of H do not have class labels and the system must determine natural partitions of the sample data.

For example, consider the application of pattern recognition to *image segmentation*, the classification of image pixels into groupings which represent some higher entity or information in the images. Unfortunately, it is rare to have either a statistical model or a training set to aid in this grouping. Therefore, so-called unsupervised learning techniques are often applied.

Two unsupervised learning approaches which embody more general measures of feature vector similarity and do not require H are known as hierarchical clustering and partitional clustering. A set of feature vectors is sequentially partitioned (or merged) on the basis of dissimilarity (or similarity). Thus, given only a similarity measure, we either aggregate feature vectors into a single class or sequentially subdivide feature vector partitions. A neural network-based example of unsupervised learning is the Kohonen self-organizing feature maps (SOFMs).

STATISTICAL PATTERN RECOGNITION (StatPR)

Statistical Analysis

StatPR is used to develop statistically-based decision or classification strategies, which form classifiers and attempts to integrate all available problem information, such as measurements and a priori probabilities. Decision rules may be formulated in several interrelated ways. A measure of expected classification error or risk may be formulated, and a decision rule is then developed which minimizes this measure. The Bayesian approach involves converting an a priori class probability $P(w_i)$ into a measurement-conditioned (a posteriori) probability $P(w_i|\boldsymbol{x})$. This leads to a partitioning of R^d, and may be implemented via discriminant functions.

Bayesian Decision Theory

In the Bayesian approach, the extracted features $\boldsymbol{x}$, are modeled as a realization of a (continuous) random vector, $\boldsymbol{X}$. The case of discrete r.v.s is treated similarly, but with probabilities, as opposed to density functions for characterization of $\boldsymbol{x}$. Suppose the class conditioned probability density functions for feature vector $\boldsymbol{x}$, that is $p(\boldsymbol{x}|w_i)$ where $i = 1, c$ are available. This may be the result of training or learning. Assume that something is known about the a priori (i.e., before measurement) likelihood of the occurrence of class w_1 or w_2, specifically assume the a priori probabilities $P(w_i)$, $i = 1, c$ are known. For example, in the shim-bolt example above, if we know that on a given day we inspect four times as many shims as bolts, then $P(w_1) = 0.8$ and $P(w_2) = 0.2$. In the absence of this information, an often reasonable assumption is that $P(w_i) = 1/c$, that is, the a priori probabilities of the states of nature are equal.

Using Bayes's Theorem. Bayes's theorem is used to enable a solution to the classification problem which uses available feature and training data. The a priori estimate of the probability of a certain class is converted to the a posteriori, or measurement conditioned, probability of a state of nature via:

$$P(w_i|\boldsymbol{x}) = \frac{[p(\boldsymbol{x}|w_i)P(w_i)]}{p(\boldsymbol{x})}$$

where

$$p(\boldsymbol{x}) = \sum_i p(\boldsymbol{x}|w_i)$$

An intuitive classification strategy is that a given realization or sample vector, $\boldsymbol{x}$, is classified by choosing the state of nature, w_i, for which $P(w_i|\boldsymbol{x})$ is largest. Notice the quantity $p(\boldsymbol{x})$ is common to all class-conditional probabilities, therefore it represents a scaling factor which may be eliminated. Thus, in our shim-bolt example, the decision or classification algorithm is:

$$\text{choose} \begin{cases} w_1 & \text{if } p(\boldsymbol{x}|w_1)P(w_1) > p(\boldsymbol{x}|w_2)P(w_2) \\ w_2 & \text{if } p(\boldsymbol{x}|w_2)P(w_2) > p(\boldsymbol{x}|w_1)P(w_1) \end{cases}$$

Note also that any monotonically nondecreasing function of $P(w_i|\boldsymbol{x})$ may be used for this test (see discriminant functions, next). The significance of this approach is that both a priori information $[P(w_i)]$ and measurement-related information $[p(\boldsymbol{x}|w_i)]$ are combined in the decision procedure. If $P(w_1) \neq P(w_2)$, for example, this information may be explicitly incorporated in the decision process.

Decision Regions and Discriminant Functions

A *classifier* partitions feature space into class-labeled decision regions. In order to use decision regions for a possible and unique class assignment, these regions must cover R^d and be disjoint (nonoverlapping). An exception to the last constraint is the notion of fuzzy sets. The border of each decision region is a decision boundary. With this viewpoint, classification of feature vector $\boldsymbol{x}$ becomes quite simple: we determine the decision region (in R^d) into which $\boldsymbol{x}$ falls, and assign $\boldsymbol{x}$ to this class. Although the classification strategy is straightforward, the determination of decision regions is a challenge. It is sometimes convenient, yet not always necessary (or possible), to visualize decision regions and boundaries. Moreover, computational and geometric aspects of certain decision boundaries (e.g., linear classifiers which generate hyperplanar decision boundaries) are noteworthy.

A number of classifiers are based upon discriminant functions. In the c-class case, discriminant functions, denoted $g_i(\boldsymbol{x})$, $i = 1, 2, \ldots c$, are used to partition R^d using the decision rule: Assign $\boldsymbol{x}$ to class w_m (region R_m), where $g_m(\boldsymbol{x}) > g_i(\boldsymbol{x})\ \forall\ i = 1, 2, \ldots c$ and $i \neq m$. The case where $g_k(\boldsymbol{x}) = g_l(\boldsymbol{x})$ defines a decision boundary.

Linear Separability. If a linear decision boundary (hyperplanar decision boundary) exists that correctly classifies all the training samples in H for a $c = 2$ class problem, the samples are said to be linearly separable. This hyperplane, denoted H_{ij}, is defined by parameters $\boldsymbol{w}$ and w_o in a linear constraint of the form

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} - w_o = 0 \tag{1}$$

$g(\boldsymbol{x})$ separates R^d into positive and negative regions R_p and R_n, where

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} - w_o = \begin{cases} > 0 & \text{if } \boldsymbol{x} \in R_p \\ 0 & \text{if } \boldsymbol{x} \in H_{ij} \\ < 0 & \text{if } \boldsymbol{x} \in R_n \end{cases} \tag{2}$$

Problems which are not linearly separable are sometimes referred to as nonlinearly separable or topologically complex.

From a pattern recognition viewpoint, the computational advantages (both in implementation and training) and ease of visualization of linear classifiers account for their popularity. Seminal works include Refs. 11, 12, and 13.

Using the Bayesian approach, one choice of discriminant function is $g_i(\boldsymbol{x}) = P(w_i|\boldsymbol{x})$. In the case of equal a priori probabilities and class-conditioned Gaussian density functions, Ref. 2 shows that the decision boundaries are hyperplanes.

Training in Statistical Pattern Recognition. One of the problems not addressed in the previous section is determination of the parameters for the class-conditioned probability density functions. A labeled set of training samples, that is, sets of labeled feature vectors with known class, are often used. This training set is denoted H. In the case of Gaussian probability distribution function (pdf) models, it is only necessary to estimate $\boldsymbol{\mu}_i$ and Σ_i for each class. Large-dimension feature vectors, and consequently density functions, lead to situations wherein this approach is impractical. For example, in an image processing application if we use the gray level measurements directly as features, an image with 100×100 pixel spatial resolution yields a 1000×1 feature vector, and requires estimation of a 1000×1000 covariance matrix. This is seldom practical.

Nearest Neighbor Classification

An alternative, which is related to the minimum distance classification approach, is the use of a nonparametric technique known as nearest neighbor classification. We illustrate the concept of a 1-nearest neighbor classification rule (1-NNR) first. Given a feature vector, $\boldsymbol{x}$, we determine the vector in H which is closest (in terms of some distance measure) to $\boldsymbol{x}$, and denote this vector $\boldsymbol{x}'$. $\boldsymbol{x}$ is classified by assigning it to the class corresponding to $\boldsymbol{x}'$. A variation is the k-NNR, where the k samples in H which are nearest to $\boldsymbol{x}$ are determined, and the class of $\boldsymbol{x}$ is based upon some measure of the labels of these samples (e.g., a voting scheme may be employed). This approach, although conceptually and computationally straightforward, may be shown to have a greater error rate than the minimum distance classifier. However, the concept of classification based upon nearness, or similarity, of features is significant.

General Decision Rules. We formulate a *loss* function, *cost* function, or *risk* function, denoted λ_{ij}, as the cost or risk of choosing class w_i when class w_j is the true class. For example, in the $c = 2$ (w_1 or w_2) case, there are four values of λ_{ij}, that is, λ_{11}, λ_{12}, λ_{21}, λ_{22}. λ_{11}, and λ_{22} are the costs (or perhaps rewards for a correct decision) whereas λ_{12} and λ_{21} are the costs of a classification error. It is desirable to measure or estimate overall classification risk. To do this, the decision rule, cost functions, the observations, and $\boldsymbol{x}$ are used. A decision or classification to choose class w_i is denoted α_i. A decision rule is a mapping of the observed feature vector, $\boldsymbol{x}$, into an α_i through a decision rule $\alpha(\boldsymbol{x})$:

$$\alpha(\boldsymbol{x}) \to \{\alpha_1, \alpha_2 \ldots \alpha_c\}$$

Since

$$P(\alpha_i \cap w_j) = P(\alpha_i|w_j)P(w_j)$$

an overall risk measure for the $c = 2$ case is

$$\begin{aligned} R = {} & \lambda_{11}P(\alpha_1|w_1)P(w_1) + \lambda_{21}P(\alpha_2|w_1)P(w_1) \\ & + \lambda_{12}P(\alpha_1|w_2)P(w_2) + \lambda_{22}P(\alpha_2|w_2)P(w_2) \end{aligned}$$

Of course, the $P(\alpha_i|w_j)$ terms depend upon the chosen mapping $\alpha(\boldsymbol{x}) \to \alpha_i$, which in turn depends upon $\boldsymbol{x}$. Thus, a measure of conditional risk associated with a $c = 2$ class decision rule is:

$$R(\alpha(\boldsymbol{x}) \to \alpha_1) = R(\alpha_1|\boldsymbol{x}) = \lambda_{11}P(w_1|\boldsymbol{x}) + \lambda_{12}P(w_2|\boldsymbol{x})$$

for α_1 and

$$R(\alpha(\boldsymbol{x}) \to \alpha_2) = R(\alpha_2|\boldsymbol{x}) = \lambda_{21}P(w_1|\boldsymbol{x}) + \lambda_{22}P(w_2|\boldsymbol{x})$$

for α_2. For a c class decision problem, the expected risk is given by an application of the total probability theorem:

$$R(\alpha(\boldsymbol{x})) = \int R(\alpha(\boldsymbol{x})|\boldsymbol{x})p(\boldsymbol{x})\,\boldsymbol{dx}$$

Minimizing the conditional risk, $R(\alpha(\boldsymbol{x})|\boldsymbol{x})$ thus minimizes the expected risk. The lower bound on $R(\alpha(\boldsymbol{x}))$ is often referred to as the *Bayes risk*. In order to minimize $\overline{R}(\alpha(\boldsymbol{x}))$ for $c = 2$, since only two choices or classifications (α_1 or α_2) are possible, the decision rule is formulated as:

$$R(\alpha_1|\boldsymbol{x}) \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} R(\alpha_2|\boldsymbol{x})$$

This may be expanded into

$$\lambda_{11}P(w_1|\boldsymbol{x}) + \lambda_{12}P(w_2|\boldsymbol{x}) \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} \lambda_{21}P(w_1|\boldsymbol{x}) + \lambda_{22}P(w_2|\boldsymbol{x})$$

or

$$(\lambda_{11} - \lambda_{21})p(\boldsymbol{x}|w_1)P(w_1) \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} (\lambda_{22} - \lambda_{12})p(\boldsymbol{x}|w_2)P(w_2)$$

When $\lambda_{11} = \lambda_{22} = 0$ (there is no cost or risk in a correct classification) and $(\lambda_{11} - \lambda_{21}) < 0$, this may be rewritten as:

$$\frac{p(\boldsymbol{x}|w_1)}{p(\boldsymbol{x}|w_2)} \underset{\alpha_1}{\overset{\alpha_2}{\gtrless}} \frac{(\lambda_{22} - \lambda_{12})}{(\lambda_{11} - \lambda_{21})} \frac{P(w_2)}{P(w_1)}$$

This form yields a classifier based upon a likelihood ratio test (LRT).

For c classes, with the loss function:

$$\lambda_{ij} = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

all errors are equally costly. The conditional risk of decision α_i is:

$$\begin{aligned} R(\alpha(\boldsymbol{x}) \to \alpha_i) &= \sum_{j=1}^{c} \lambda_{ij}P(w_j|\boldsymbol{x}) \\ &= \sum_{j\neq 1} P(w_j|\boldsymbol{x}) = 1 - P(w_i|\boldsymbol{x}) \end{aligned}$$

To minimize the conditional risk, the decision rule is therefore to choose the α_i which maximizes $P(w_i|\boldsymbol{x})$, that is, the w_i for which $P(w_i|\boldsymbol{x})$ is largest. This is intuitively appealing. Since $P(w_i|\boldsymbol{x})$ is the a posteriori probability, this results in the maximum a posteriori probability (MAP) classifier, which may be formulated as:

$$P(w_i|\boldsymbol{x}) \overset{\alpha_i}{>} P(w_j|\boldsymbol{x}) \qquad \forall j \neq i$$

As before, Bayes's rule is used to reformulate these tests in terms of class-conditioned density functions and a priori probabilities.

For general formulations of risk (through λ_{ij}), the resulting decision rule is:

$$R(\alpha_i|\boldsymbol{x}) \overset{\alpha_i}{<} R(\alpha_j|\boldsymbol{x}) \qquad \forall i \neq j$$

Clustering

In some cases, a training set, H, is not available for a PR problem. Instead, an unlabeled set of typical features, denoted H_u is available. For each sample, $\boldsymbol{x} \in H_u$, the class origin or label is unknown. Desirable attributes of H_u are that the cardinality of H_u is large, all classes are represented in H_u, and subsets of H_u may be formed into natural groupings or clusters. Each cluster most likely (or hopefully) corresponds to an underlying pattern class.

Clustering is a popular approach in unsupervised learning (14). Clustering applications in image analysis, for example, include (15) and (16). Iterative algorithms involving cluster splitting and merging in image analysis are shown in Ref. 1.

Unsupervised learning approaches attempt to develop a representation for the given sample data, after which a classifier is designed. In this context, clustering may be conceptualized as "how do I build my fences?" Thus, in unsupervised learning, the objective is to define the classes. A number of intuitive and practical approaches exist to this problem. For example, a self-consistent procedure is:

1. Convert a set of unlabeled samples, H_u into a tentative training set, H_T.
2. Using H_T, apply a supervised training procedure and develop corresponding discriminant functions/decision regions.
3. Use the results of step 2 on H_u, that is, reclassify H_u. If the results are consistent with H_T, stop, otherwise go to 1 and revise H_T.

This approach clusters data by observing similarity. There exist neural networks with this feature (see SELF-ORGANIZING FEATURE MAPS). In many PR applications involving unsupervised learning, features naturally fall into natural, easily observed groups. In others, the grouping is unclear and very sensitive to the measure of similarity used. The c-means algorithm (2) and its derivatives are one of the most popular approaches.

The c-means algorithm:

1. Choose the number of classes, c.
2. Choose class means or exemplars, denoted $\hat{\boldsymbol{\mu}}_i$, $\hat{\boldsymbol{\mu}}_2$, . . . $\hat{\boldsymbol{\mu}}_c$.
3. Classify each of the unlabeled samples, $\boldsymbol{x}_k$ in H_u.
4. Recompute the estimates for $\hat{\boldsymbol{\mu}}_i$, using the results of step 3.
5. If the $\hat{\boldsymbol{\mu}}_i$, are consistent, stop, otherwise go to step 1, 2, or 3.

Notice the essence of this approach is to achieve a self-consistent partitioning of the data. Choice of initial parameters [c and $\boldsymbol{\mu}_i(0)$] is a challenging issue. This spawns an area of study concerning cluster validity.

An Example of the c-Means Algorithm. Figure 3 shows examples of the c-means algorithm for the $c = 2$ class case on a set of unlabeled data. The trajectory of the $\boldsymbol{\mu}_i$, as a function of iteration is shown.

Iterative and Hierarchical Clustering. Clustering may be achieved through a number of alternative strategies, including iterative and hierarchical approaches. Hierarchical strategies may further be subdivided into agglomerative (merging of clusters) or devisive (splitting of clusters). Hierarchical strategies have the property that not all partitions of the data

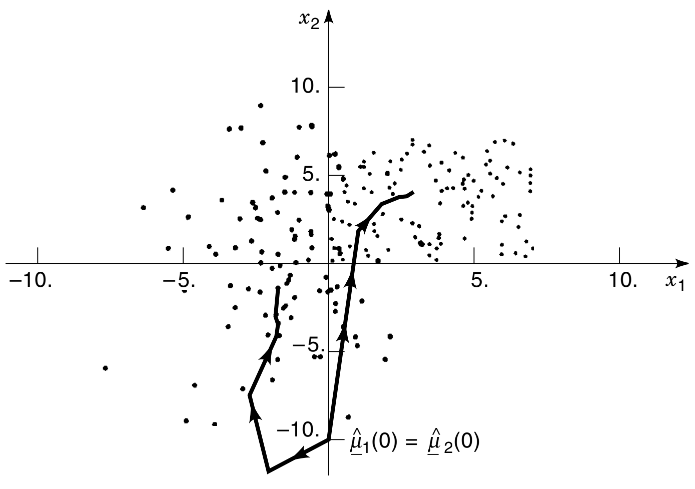

Figure 3. Example of the trajectories of the class means in the c-means algorithm [from (2) Copyright 1992. Reprinted by permission of John Wiley & Sons.]

are considered. However, when the number of samples is large, hierarchical clustering may be inappropriate. In an agglomerative procedure, two samples, once in the same class, remain in the same class throughout subsequent cluster merging. This may lead to resulting data partitions being suboptimal.

Clustering Criterion Functions. Developing appropriate similarity measures $d(\boldsymbol{x}_i, x_j)$ is paramount in clustering. For a given partition of H_u, denoted P, a measure of the goodness of the overall clustering is given by clustering criterion function, $J(P)$. If

$$J(P_1) < J(P_2)$$

P_1 is a better partition than P_2. Once a suitable $J(P)$ is defined, the objective is to find P_m such that

$$J(P_m) = P^{\min}(J(P))$$

in a computationally efficient manner. This is a problem in discrete optimization. One of the more popular clustering metrics is the sum of squared error (SSE) criterion. Given n_i samples in H_i, with sample mean $\boldsymbol{m}_i$, where

$$\boldsymbol{m}_i = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in H_i} \boldsymbol{x}_j$$

the SSE criterion, J_{SSE} is defined as

$$J_{\mathrm{SSE}}(P) = \sum_{i=1}^{c} \sum_{\boldsymbol{x}_j \in H_i} \|\boldsymbol{x} - \boldsymbol{m}_i\|^2$$

J_{SSE} thus indicates the total variance for a given partition. For example, cluster-swapping approaches are a variant on the c-means iterative algorithm which implement a good cluster reorganization strategy, where good means

$$J_{\mathrm{SSE}}(P_{k+1}) \leq J_{\mathrm{SSE}}(P_k)$$

For illustration, our reorganization strategy is restricted to the movement of a single vector, $\boldsymbol{x}_j$ from H_i to H_j, denoted $H \overset{x_j}{\to} H_j$. The revised clusters in P_{k+1} are denoted H_i and H_j. It is possible to show $H_i \overset{x_k}{\to} H_j$ decreases $J_{\text{SSE}}(P_k)$ if

$$\left(\frac{n_j}{n_j+1}\right)\|\boldsymbol{x}_j-\boldsymbol{m}_j\|^2 < \left(\frac{n_i}{n_i-1}\right)\|\boldsymbol{x}_j-\boldsymbol{m}_i\|^2$$

Hierarchical Clustering. Consider a hierarchical clustering procedure in which clusters are merged so as to produce the smallest increase in the sum-of-squared error at each step. The ith cluster or partition, denoted H_i, contains n_i samples with sample mean m_i. The smallest increase results from merging the pair of clusters for which the measure M_{ij}, where

$$M_{ij} = \frac{n_i n_j}{n_i+n_j}\|\boldsymbol{m}_i-\boldsymbol{m}_j\|^2$$

is minimum. Recall

$$J_e = \sum_{i=1}^{c}\sum_{\boldsymbol{x}\in H_i}\|\boldsymbol{x}-\boldsymbol{m}_i\|^2$$

that is, J_e measures the total squared error incurred in representing the n samples $\boldsymbol{x}_1, \ldots \boldsymbol{x}_n$ by c cluster means $\boldsymbol{m}_1 \ldots \boldsymbol{m}_c$.

The change in the SSE after merging clusters i and j is

$$\Delta J_e = -\left(\sum_{\boldsymbol{x}\in H_i}\|\boldsymbol{x}-\boldsymbol{m}_i\|^2 + \sum_{\boldsymbol{x}\in H_j}\|\boldsymbol{x}-\boldsymbol{m}_j\|^2\right) + \sum_{\boldsymbol{x}\in H_i \text{ or } H_j}\|\boldsymbol{x}-\boldsymbol{m}_{ij}\|^2$$

where

$$\boldsymbol{m}_i = \frac{1}{n_i}\sum_{\boldsymbol{x}\in H_i}\boldsymbol{x} \quad \boldsymbol{m}_j = \frac{1}{n_j}\sum_{\boldsymbol{x}\in H_j}\boldsymbol{x}$$

and

$$\boldsymbol{m}_{ij} = \frac{1}{n_i+n_j}\sum_{\boldsymbol{x}\in H_i \text{ or } H_j}\boldsymbol{x}$$

The objective is to merge clusters such that ΔJ_e is minimum. It is possible to show:

$$\Delta J_e = \frac{n_i n_j}{(n_i+n_j)}\|\boldsymbol{m}_j-\boldsymbol{m}_i\|^2 = \frac{n_i n_j}{(n_i+n_j)}\|\boldsymbol{m}_i-\boldsymbol{m}_j\|^2$$

and therefore use this measure in choosing clusters to merge.

The popularity of clustering has spawned a sizable and varied library of clustering algorithms and software (17), one of the most popular being the *ISODATA (Iterative Self-Organizing Data Analysis Techniques A) algorithm* (10,18).

SYNTACTIC (STRUCTURAL) PATTERN RECOGNITION

Many times the significant information in a pattern is not merely in the presence or absence, or the numerical values, of a set of features. Instead, the interrelationships or interconnections of features yield important structural information, which facilitates structural description or classification. This is the basis of syntactic (or structural) pattern recognition. Figure 4 shows the general strategy.

In using SyntPR approaches, it is necessary to quantify and extract structural information and determine the structural similarity of patterns. One syntactic approach is to relate the structure of patterns with the syntax of a formally defined language, in order to capitalize on the vast body of knowledge related to pattern (sentence) generation and analysis (parsing). Syntactic pattern recognition approaches are presented in Refs. 2, 19, 20, 21, 22, and 23. A unified view of StatPR and SyntPR is shown in Ref. 24. An extended example of the utility of SyntPR in an image interpretation application appears in Ref. 25.

Typically, SyntPR approaches formulate hierarchical descriptions of complex patterns built up from simpler subpatterns. At the lowest level, primitive elements or building blocks are extracted from the input data. One distinguishing characteristic of SyntPR involves the choice of primitives. Primitives must be subpatterns or building blocks, whereas features (in StatPR) are any measurements.

Syntactic structure quantification is shown using two approaches: formal grammars, and relational descriptions (attributed graphs). These tools allow structurally quantitative pattern representation, which facilitate recognition, classification, or description. A class of procedures for syntactic recognition, including parsing (for formal grammars) and relational graph matching (for attributed relational graphs) are then developed. While it is not mandatory, many SyntPR techniques are based upon generation and analysis of complex patterns by a hierarchical decomposition into simpler patterns.

Formal Grammars and Syntactic Recognition by Parsing

The syntax rules of formal grammars may be used to generate patterns (possibly from other patterns) with constrained structural relations. A grammar may therefore serve to model a class-specific pattern-generating source which generates all the patterns with a class-specific structure. Furthermore, it is desirable to have each class-specific grammar derivable from a set of sample patterns, that is, training must be considered. This raises the issue of grammatical inference.

Useful introductions to formal grammars appear in Refs. 26 and 27. References 19, 21, 22, and 23 are devoted entirely to SyntPR.

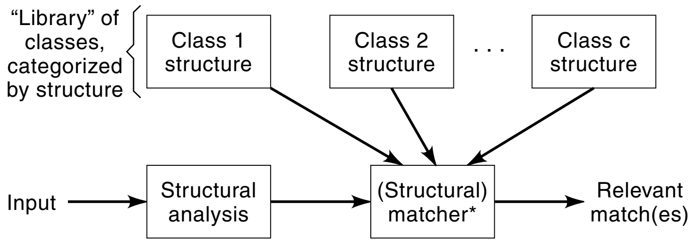

Figure 4. Generic syntactic (or structural) pattern recognition system [from (2) Copyright 1992. Reprinted by permission of John Wiley & Sons.]

Grammars. A *grammar* consists of the following four entities:

1. A set of terminal or primitive symbols (primitives), denoted V_T (or, alternately, Σ). In many applications, the choice of the terminal set or primitives is difficult, and has a large component of art, as opposed to science.
2. A set of non-terminal symbols, or variables, which are used as intermediate quantities in the generation of an outcome consisting solely of terminal symbols. This set is denoted as V_N (or, alternately, N).
3. A set of productions, or production rules or rewriting rules which allow the previous substitutions. It is this set of productions, coupled with the terminal symbols, which principally gives the grammar its structure. The set of productions is denoted P.
4. A starting (or root) symbol, denoted S. $S \in V_N$.

Note that V_T and V_N are disjoint sets, that is, $V_T \cap V_N = \emptyset$.

Thus, using the above definitions, we formally denote a grammar, G, as the four-tuple:

$$G = (V_T, V_N, P, S)$$

Constraining Productions. Given V_T and V_N, the productions, P, may be viewed as constraints on how class-specific patterns may be described. Different types of grammars place restrictions on these mappings. For example, it is reasonable to constrain elements of P to the form

$$A \to B$$

where

$$A \in (V_N \cup V_T)^+ - V_T^+$$

and

$$B \in (V_N \cup V_T)^*$$

Thus, A must consist of at least one member of V_N (i.e., a nonterminal), and B is allowed to consist of any arrangement of terminals and nonterminals. This is a partial characterization of phrase structure grammar.

Grammar Application Modes. A grammar may be used in one of two modes: Generative, in which the grammar is used to create a string of terminal symbols using P; a sentence in the language of the grammar is thus generated.

Analytic, a sentence (possibly in the language of the grammar), together with specification of G, one seeks to determine: If the sentence was generated by G; and, if so, the structure (usually characterized as the sequence of productions used) of the sentence.

The following formal notation is used. Symbols beginning with a capital letter (e.g., S_1 or S) are elements of V_N. Symbols beginning with a lowercase letter (e.g., a or b) are elements of V_T. n denotes the length of string s, that is,

$$n = |s|$$

Greek letters (e.g., α, β) represent (possibly empty) strings, typically comprised of terminals and/or nonterminals.

Constraints on the production or rewrite rules, P, in string grammar G are explored by considering the general production form:

$$\alpha_1 \to \beta_2$$

which means string α_1 is replaced by string β_2. In general, α_1 and β_2 may contain terminals and/or nonterminals.

In a context-free grammar, the production restrictions are:

$$\alpha_1 = S_1 \in V_N$$

that is, α_1 must be a single nonterminal for every production in P, and

$$|S_1| \leq |\beta_2|$$

An alternate characterization of a T_2 grammar is that every production must be of the form:

$$S_1 \to \beta_2$$

where $\beta_2 \in (V_N \cup V_T)^* - \{\epsilon\}$. Note the restriction in the above productions to the replacement of S_1 by string β_2 independently of the context in which S_1 appears.

Context-free grammars can generate a string of terminals and/or nonterminals in a single production. Moreover, since productions of the form $A \to \alpha A \beta$ are allowed, context-free grammars are self-embedding.

Context-free grammars are important because they are the most descriptively versatile grammars for which effective (and efficient) parsers are available. The production restrictions increase in going from context-sensitive to context-free grammars.

Finite-state (FS) or regular grammars are extremely popular. The production restrictions in a finite-state or regular grammar are those of a context-free grammar, plus the additional restriction that at most one nonterminal symbol is allowed on each side of the production. That is,

$$\alpha_1 = S_1 \in V_N$$
$$|S_1| \leq |\beta_2|$$

and productions are restricted to:

$$A_1 \to a$$

or

$$A_1 \to aA_2$$

Finite-state (FS) grammars have many well-known characteristics which explain their popularity, including simple graphical representations and known tests for equivalence. Finite state grammars are useful when analysis (parsing) is to be accomplished with finite-state machines (26).

Other Grammar Types Used for Syntactic Pattern Recognition. Grammars other than string grammars exist and are usually distinguished by their terminals and nonterminals (as opposed to constraints on P). These are useful in 2-D and higher dimensional pattern representation applications, in that the structure of the productions involving terminals and nonterminals is greater than one dimensional. Higher dimensional grammars also facilitate relational descriptions. Productions in higher dimensional grammars are usually more complex, since rewriting rules embody operations more complex than simple 1-D string rewriting. For example, in 2-D cases standard attachment points are defined. Two of the more popular are tree grammars and web grammars (19). Not surprisingly, there is little correlation between the dimension of the grammar used for pattern generation and the dimensionality of the pattern space. For example, a 1-D grammar may be used for 2-D or 3-D patterns.

Example of Grammatical Pattern Description for Chromosome Classification. Figure 5, excerpted from Ref. 28, shows the conversion of a chromosome outline to a string in a formal grammar, where the primitives and productions are given. Using the primitives and productions of grammar, G_M, given in Fig. 5(a), the string $x = cbbabbbbdbbbbabbbcbbbabbbbd$ $bbbbabbb$ may be produced to describe the sample chromosome outline shown in Fig. 5(b).

Parsing

Chomsky Normal Form. A CFG is in Chomsky normal form (CNF) if each element of P is in one of the following forms:

$$A \rightarrow BC \text{ where } A, B, C \in V_N$$
$$A \rightarrow a \text{ where } A \in V_N,\ a \in V_T$$

The Cocke-Younger-Kasami (CYK) Parsing Algorithm. The CYK algorithm is a parsing approach which will parse string x in a number of steps proportional to $|x|^3$. The CYK algorithm requires the CFG be in Chomsky normal form (CNF). With this restriction, the derivation of any string involves a series of binary decisions. First, the CYK table is formed. Given string $x = x_1, x_2, \ldots x_n$, where $x_i \in V_T$, $|x| = n$, and a grammar, G, we form a triangular table with entries t_{ij} indexed by

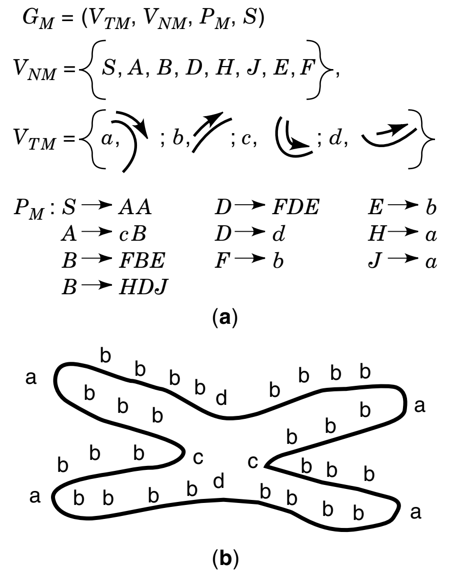

Figure 5. Conversion of a chromosome outline to a string in a formal grammar [excerpted from (28) Copyright 1972, IEEE]. (a) Primitives and productions in L(G). (b) Sample chromosome outline yielding string $x = cbbbabbbbdbbbbabbbcbbbabbbbdbbbbabbb$.

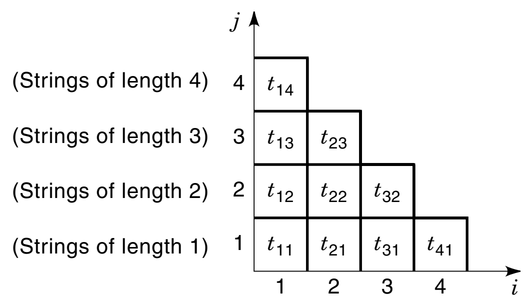
j
(Strings of length 4) 4 t_{14}
(Strings of length 3) 3 t_{13} t_{23}
(Strings of length 2) 2 t_{12} t_{22} t_{32}
(Strings of length 1) 1 t_{11} t_{21} t_{31} t_{41}
1 2 3 4 i

Figure 6. Structure of CYK parse table [from (2) Copyright 1992. Reprinted by permission of John Wiley & Sons.]

i and j where $1 \le i \le n$ and $1 \le j \le (n - i + 1)$. The origin is at $i = j = 1$, and entry t_{11} is the lower-left hand entry in the table. t_{1n} is the uppermost entry in the table. This structure is shown in Fig. 6.

To build the CYK table, a few simple rules are used. Starting from location (1, 1), if a substring of $\boldsymbol{x}$, beginning with $\boldsymbol{x}_i$, and of length $\boldsymbol{j}$ can be derived from a nonterminal, this nonterminal is placed into cell $(\boldsymbol{i}, \boldsymbol{j})$. If cell $(1, n)$ contains S, the table contains a valid derivation of x in $L(G)$. It is convenient to list the x_i, starting with $i = 1$, under the bottom row of the table.

Example. Sample use of grammars and the CYK parsing algorithm for recognition:

Sample Grammar Productions. These are shown next. With these constraints, notice there are six forms for the derivation of the string $x = aabb$.

$$S \rightarrow AB|BB$$
$$A \rightarrow CC|AB|a$$
$$B \rightarrow BB|CA|b$$
$$C \rightarrow BA|AA|b$$

Parse Table for String x = aabb. Construction of an example parse table is shown in Fig. 7. Recall cell entry (i, j) corresponds to the possibility of production of a string of length j, starting with symbol x_i. The table is formed from the bottom row ($j = 1$) upwards. Entries for cells (1, 1), (2, 1), (3, 1), and (4, 1) are relatively easy to determine, since they each correspond to production of a single terminal. For the second ($j = 2$) row of the table, all nonterminals which could yield deriva-

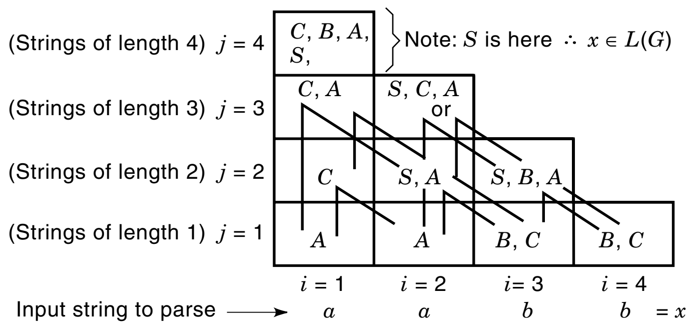
(Strings of length 4) $j = 4$ $C, B, A, S,$
Note: S is here $\therefore x \in L(G)$
(Strings of length 3) $j = 3$ C, A S, C, A or
(Strings of length 2) $j = 2$ C S, A S, B, A
(Strings of length 1) $j = 1$ A A B, C B, C
$i = 1$ $i = 2$ $i = 3$ $i = 4$
Input string to parse a a b b $= x$

Figure 7. Construction of a sample parse table for the string $x = aabb$ [from (2) Copyright 1992. Reprinted by permission of John Wiley & Sons.]

tions of substrings of length 2, beginning with x_i $i = 1, 2, 3$, must be considered. For example, cell (1, 2) corresponds to production of two-terminal long-string beginning with a. Alternately, it is only necessary to consider nonterminals which produce AA, as shown in the $j = 1$ row of the table. From Fig. 7, only nonterminal C, in the production $C \rightarrow BA|AA|b$ satisfies this.

Forming the third and fourth ($j = 3$ and $j = 4$ respectively) rows of the table is slightly more complicated. For example, cell (1, 3) corresponds to strings of length 3, beginning with terminal x_1 (a) in this case. This requires examination of cells (1, 1) and (2, 2), corresponding to producing the desired string with 1 nonterminal followed by 2 nonterminals, (denoted {1 + 2} hereafter) as well as cells (1, 2) and (3, 1) (denoted the {2 + 1} derivation). For the former, it is necessary to consider production of AS, and AA, and nonterminal C is applicable. For the latter, the production of CB and CC is considered, yielding A. Thus, cell (1, 3) contains nonterminals C and A. Similarly, for cell (2, 3), cells (2, 1) and (3, 2) (the {1 + 2} derivation) as well as (2, 2) and (4, 1) (the {2 + 1} derivation) must be considered.

Finally, formation of cell (1, 4) is considered. Possible cell pairings to consider are summarized here

$$(1, 1) \text{ and } (2, 3)\ \{1 + 3\} \rightarrow AS, AC, \underline{AA} : C$$
$$(1, 2) \text{ and } (3, 2)\ \{2 + 2\} \rightarrow CS, CB, \underline{CA} : B$$
$$(1, 3) \text{ and } (4, 1)\ \{3 + 1\} \rightarrow CB, \underline{CC}, \underline{AB}, AC : A, S$$

Cell pairings which yield a possible nonterminal are shown underlined. Thus, (1, 4) contains nonterminals C, B, A, S. Since this includes the starting symbol, the parse succeeds and $aabb$ is a valid string in the language of this grammar. Note that since the grammar is in CNF, it is never necessary to consider more than two-cell pairings (although as we increase j the number of possible pairings increases).

String Matching. A somewhat simpler approach to classification or recognition of entities using syntactic descriptions is a matching procedure. Consider the c class case. Class-specific grammars $G_1, G_2, \ldots G_c$ are developed. Given an unknown description, x, to classify, it is necessary to determine if $x \in L(G_i)$ for $i = 1, 2, \ldots c$. Suppose the language of each G_i could be generated and stored in a class-specific library of patterns. By matching x against each pattern in each library, the class membership of x could be determined. String matching metrics yield classification strategies which are a variant of the 1-NNR rule for feature vectors, where a matching metric using strings instead of vectors is employed.

There are several shortcomings to this procedure. First, often $|L(G_i)| = \infty$, therefore the cataloging or library based procedure is impossible. Second, even if $L(G_i)$ for each i is denumerable, it usually requires very large libraries. Consequently, the computational effort in matching is excessive. Third, it is an inefficient procedure. Alternatives which employ efficient search algorithms, prescreening of the data, the use of hierarchical matching and prototypical strings are often preferable. Note that in SyntPR, the similarity measure(s) used must account for the similarity of primitives as well as similarity of structure.

Graphical Approaches Using Attributed Relational Graphs

Digraphs and Attributed Relational Graphs (ARG). Directed graphs or digraphs are valuable tools for representing relational information. Here we represent graph G as $G = \{N, R\}$ where N is a set of nodes (or vertices) and R is a subset of $N \times N$, indicating arcs (or edges) in G.

In addition to representing pattern structure, the representation may be extended to include numerical and perhaps symbolic attributes of pattern primitives (i.e., relational graph nodes). An extended representation includes features or properties as well as relations with other entities. An attributed graph, as defined next, results.

Attributed Graphs. An attributed graph, G_i, is a 3-tuple and is defined as follows:

$$G_i = \{N_i, P_i, R_i\}$$

where N_i is a set of nodes, P_i is a set of properties of these nodes, and R_i is a set of relations between nodes. (An alternative viewpoint is that R_i indicates the labeled arcs of G_i, where if an arc exists between nodes a and b, then R_i contains element (a, b).)

Attributed Relational Graph Example: Character Recognition. Figure 8 (courtesy of R. D. Ferrell) shows an example of ARGs used to quantify the structure of block characters C and L. Each line segment of the character is an attributed node in the corresponding graph, with a single attribute indicating either horizontal or vertical spatial orientation. Node relations used indicate whether the segments meet at a 90° or 180° angle, as well as connectedness above or to the left.

Comparing Attributed Relational Graphs. One way to recognize structure using graphs is to let each pattern (structural) class be represented by a prototypical relational graph. An

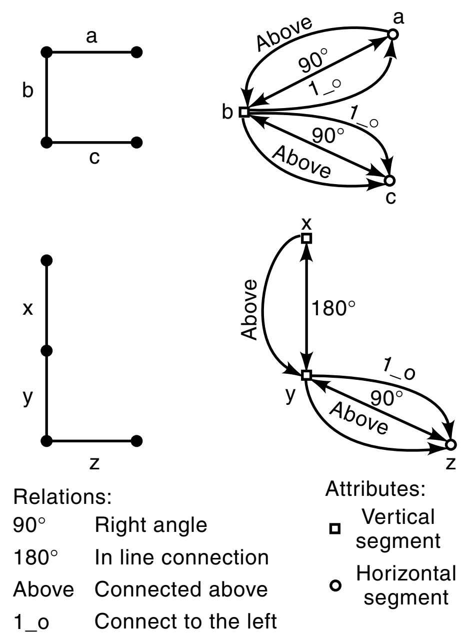

Figure 8. Example of ARGs used to quantify the structure of block characters.

unknown input pattern is then converted into a structural representation in the form of a representational graph and this graph is then compared with the relational graphs for each class. Notice that compared does not necessarily mean matched verbatim.

Attributed Relational Graph Matching Measures which Allow Structural Deformations. In order to allow structural deformations, numerous match or distance measures have been proposed. These include (29,30):

- Extraction of features from G_1 and G_2, thereby forming feature vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. This is followed by the use of StatPR techniques to compare $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Note the features are graph features, as opposed to direct pattern features.
- Using a matching metric the minimum number of transformations necessary to transform $\boldsymbol{G}_1$ (the input) into $\boldsymbol{G}_2$ (the reference). Common transformations include: node insertion, node deletion, node splitting, node merging, vertex insertion, and vertex deletion.

Graph Transformation Approaches. Here we consider a set of comparisons, transformations and associated costs in deriving a measure $D(G_i, G_j)$. Desirable attributes of $D(G_i, G_j)$ are:

1. $D(G_i, G_j) = 0$
2. $D(G_i, G_j) > 0$ if $i \neq j$
3. $D(G_i, G_j) = D(G_j, G_i)$
4. $D(G_i, G_j). \leq D(G_i, G_k). + D(G_k, G_j).$

Property 4 is referred to as the triangle inequality. Property 3 requires $w_{ni} = w_{nd}$ and $w_{ei} = w_{ed}$ where w_{ni} is the cost of node insertion, w_{nd} is the cost of node deletion, w_{ei} is the cost of edge insertion, and w_{ed} is the cost of edge deletion.

Node Matching Costs and Overall Cost in Matching Attributed Relational Graphs. Since nodes possess attributes and therefore even without considering relational constraints all nodes are not equal, a similarity measure between node p_i of G_i and node q_j of G_j is required. Denote this cost $f_n(p_i, q_j)$. For candidate match between G_1 and G_2, denoted x, with p nodes, the total cost is

$$c_n(x) = \sum f_n(p_i, q_j)$$

where the summation is overall corresponding node pairs, under node mapping x. For a candidate match configuration (i.e., some pairing of nodes and subsequent transformations) the overall cost for configuration x is

$$D_S(x) = w_{ni}c_{ni} + w_{nd}c_{nd} + w_{bi}c_{bi} + w_{bd}c_{bd} + w_n c_n(x)$$

and the distance measure, D, is defined as

$$D = \overset{\min}{x}\{D_s(x)\}$$

NEURAL PATTERN RECOGNITION

Modern digital computers do not emulate the computational paradigm of biological systems. The alternative of neural computing emerged from attempts to draw upon knowledge of how biological neural systems store and manipulate information. This leads to a class of artificial neural systems termed neural networks and involves an amalgamation of research in many diverse fields such as psychology, neuroscience, cognitive science, and systems theory. Artificial neural networks (ANNs) are a relatively new computational paradigm, and it is probably safe to say that the advantages, disadvantages, applications, and relationships to traditional computing are not fully understood. Neural networks are particularly well suited for some pattern association applications.

Fundamental neural network architecture and application Refs. are 2, 31, 32, 33, 34. Rosenblatt (35) is generally credited with initial perceptron research. The general feedforward structure is also an extension of the work of Minsky/Papert (36) and the early work of Nilsson (37) on the transformations enabled by layered machines, as well as the effort of Widrow/Hoff (38) in adaptive systems. A comparison of standard and neural classification approaches is found in Ref. 39.

ANN Components

Basically, three entities characterize an ANN:

1. The network topology, or interconnection of neural units
2. The characteristics of individual units or artificial neurons
3. The strategy for pattern learning or training

As in the SyntPR and StatPR approaches to pattern recognition, the success of the NeurPR approach is likely to be strongly influenced by the quality of the training data and algorithm. Furthermore, existence of a training set and a training algorithm does not guarantee that a given network will train or generalize correctly for a specific application.

Key Aspects of Neural Computing

The following are key aspects of neural computing. The overall computational model consists of a variable interconnection of simple elements, or units. Modifying patterns of interelement connectivity as a function of training data is the key learning approach. In other words, the system knowledge, experience, or training is stored in the form of network interconnections.

To be useful, neural systems must be capable of storing information (trainable). Neural PR systems are trained with the hope that they will subsequently display correct generalized behavior when presented with new patterns to recognize or classify. That is, the objective is for the network (somehow) in the training process to develop an internal structure which enables it to correctly identify or classify new similar patterns.

Many open questions regarding neural computing, and its application to PR problems, exist. Furthermore, the mapping of a PR problem into the neural domain, that is, the design of a problem-specific neural architecture, is a challenge which requires considerable engineering judgment. A fundamental problem is selection of the network parameters, as well as the selection of critical and representable problem features.

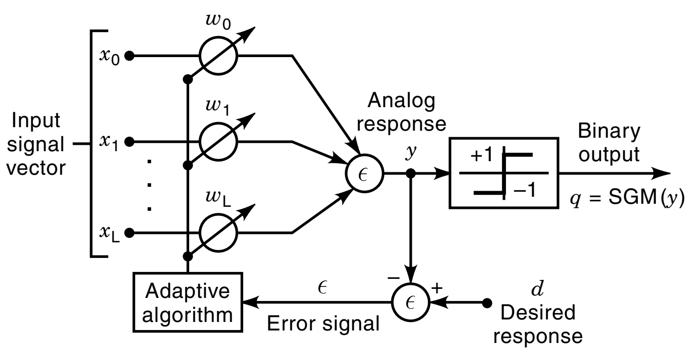

Figure 9. Basic Perceptron/Adaline element [from (41) Copyright 1988, IEEE].

Neural Network Structures for Pattern Recognition. Several different generic neural network structures, are useful for a class of pattern recognition (PR) problems. Examples are:

The Pattern Associator (PA). This neural implementation is exemplified by feedforward networks. The most commonly used learning (or training) mechanism for feedforward (FF) networks is the backpropagation approach using the generalized delta rule.

The Content-Addressable or Associative Memory Model (CAM or AM). This neural network structure is best exemplified by the recurrent network often referred to as the Hopfield model. Typical usage includes recalling stored patterns when presented with incomplete or corrupted initial patterns (see Hopfield networks).

Self-Organizing Networks. These networks exemplify neural implementations of unsupervised learning in the sense that they typically cluster, or self-organize input patterns into classes or clusters based upon some form of similarity.

Perceptrons

Perceptron and Adaline Unit Structure. The *perceptron* is a regular feedforward network layer with adaptable weights and hardlimiter activation function. Rosenblatt (35) is generally credited with initial perceptron research. The efforts of Widrow and Hoff in adaptive systems, specifically the Adaline (Adaptive Linear Element) and modified Adaline (Madeline) structures presented in Refs. 40 and 41 are also relevant. For brevity, we will consider them as one generic structure.

The units in the perceptron form a linear threshold unit, linear because of the computation of the activation value (inner product) and threshold to relate to the type of activation function (hardlimiter). Training of a perceptron is possible with the perceptron learning rule. As shown in Fig. 9, the basis for the perceptron adaline element is a single unit whose net activation is computed using

$$\text{net}_i = \sum_j w_{ij} x_j = \boldsymbol{w}^T \boldsymbol{x} \tag{3}$$

The unit output is computed by using a hard limiter, threshold-type nonlinearity, namely the signum function, i.e., for unit i with output o_i:

$$o_i = \begin{cases} +1 & \text{if} \quad \text{net}_i \geq 0 \\ -1 & \text{if} \quad \text{net}_i < 0 \end{cases} \tag{4}$$

The unit has a binary output, however the formation of net_i (as well as weight adjustments in the training algorithm) is based upon the linear portion of the unit, that is, the mapping obtained prior to application of the nonlinear activation function.

Combination of Perceptrons of Adaline Units to Achieve More Complex Mappings. Layers of Adaline units, often referred to as multilayer perceptrons or MLPs may be used to overcome the problems associated with nonlinearly separable mappings. One of the biggest shortcomings of MLPs, however, is the availability of suitable training algorithms. This shortcoming often reduces the applicability of the MLP to small, hand-worked solutions. As shown in Fig. 10, combinations of Adaline units yield the Madaline (modified Adaline) or multilayered perceptron structure, which may be used to form more complex decision regions.

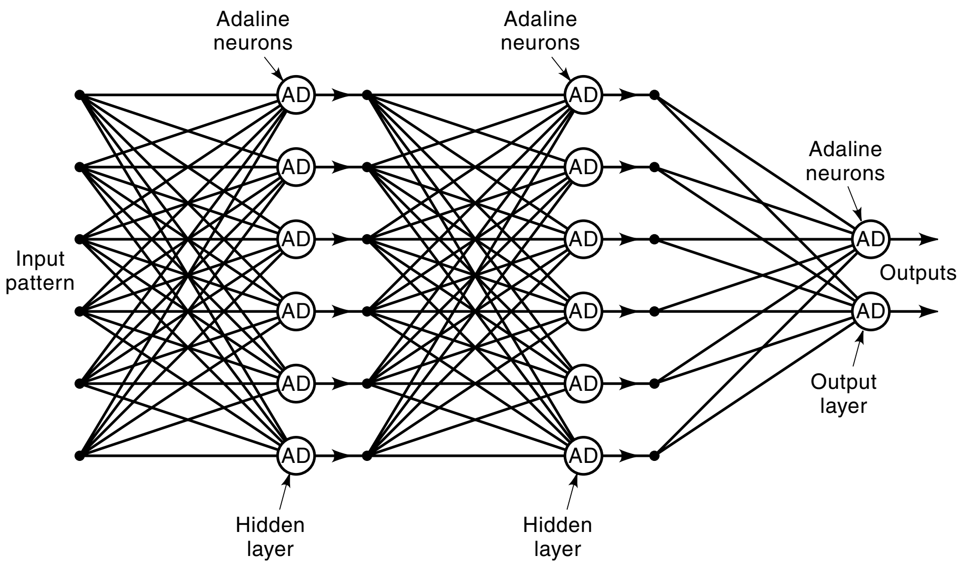

Figure 10. Using combinations of adaline units yield the multilayered perceptron (Madaline) [from (41) Copyright 1988, IEEE].

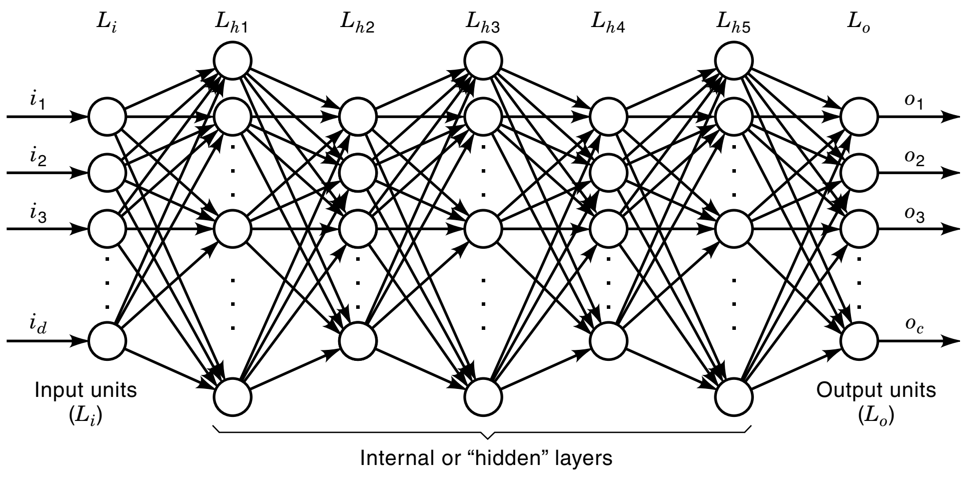

Figure 11. The typical feedforward network, consisting of layers of simple units. [from (2)].

Feedforward Networks

The feedforward network is in some sense an extension of the Madeline/perceptron structure composed of a hierarchy of processing units, organized in a series of two or more mutually exclusive sets of neurons or layers. The first, or input layer, serves as a holding site for the values applied to the network. The last, or output, layer is the point at which the final state of the network is read. Between these two extremes lie zero or more layers of hidden units; it is here that the real mapping or computing takes place. Links, or weights, connect each unit in one layer to only those in the next-higher layer. There is an implied directionality in these connections, in that the output of a unit, scaled by the value of a connecting weight, is fed forward to provide a portion of the activation for the units in the next-higher layer. Figure 11 illustrates the typical feedforward network. The network as shown consists of a layer of d input units, (L_i), a layer of c output units, (L_o), and a variable number (5 in this example) of internal or hidden layers (L_{h_i}) of units. Observe the feedforward structure, where the inputs are directly connected to only units in L_o, and the outputs of layer L_k units are only connected to units in layer L_{k+1} or are outputs, if $L_k = L_o$.

Training Feedforward Networks. Once an appropriate network structure is chosen much of the effort in designing a neural network for PR concerns the design of a reasonable training strategy. Often, for example, while observing a particular training experiment, the designer will notice the weight adjustment strategy favoring particular stimulus-response (S-R) patterns, becoming painfully slow (perhaps while stuck in a local minimum), becoming unstable, or oscillating between solutions. This necessitates engineering judgement in considering the following training parameters:

- Train by pattern or epoch
- Use of momentum and corresponding weight
- Learning weight/weight changes over time
- Sequential versus random ordering of training vectors
- Whether the training algorithm is stuck at a local energy minimum
- Suitable unit biases (if applicable)
- Appropriate initial conditions on biases, weights, and so on

Back Propagation—A Multistep Procedure for Training Feedforward Networks. Beginning with an initial (possibly random) weight assignment for a 3-layer feedforward network, proceed as follows:

Step 1. Present input $\boldsymbol{x}^p$, form outputs, o_i, of all units in network.

Step 2. Update w_{ji} for output layer.

Step 3. Update w_{ji} for hidden layer(s).

Step 4. Stop if updates are insignificant or error is below a preselected threshold, otherwise proceed to Step 1.

This leads to an adjustment scheme based upon back propagation, commonly referred to as the generalized delta rule (GDR). A summary of the GDR equation is given in Table 1.

Hopfield (Recurrent) Networks for Pattern Recognition

Hopfield (42,43) characterized a neural computational paradigm for using a neural net as an auto-associative memory. The following variables are defined:

o_i: the output state of the ith neuron

o_i: the activation threshold of the ith neuron

w_{ij}: the interconnection weight, that is, the strength of the connection from the output of neuron j to neuron i.

Thus, $\Sigma_j w_{ij} o_j$ is the total input or activation (net_i) to neuron i. Typically, $w_{ij} \in R$, although other possibilities (e.g., binary interconnections) are possible. With the constraints developed here, for a d-unit network there are $d(d-1)/2$ possibly nonzero and unique weights.

Table 1. Summary of the GDR Equations for Training Using Backpropagation

(pattern) error measure:	$E^p = \frac{1}{2}\sum_j (t_j^p - o_j^p)^2$
(pattern) weight correction	$\Delta^p w_{ji} = \varepsilon \delta_j^p \tilde{o}_i^p$
(output units)	$\delta_j^p = (t_j^p - o_j^p) f_j'(\text{net}_j^p)$
(internal units)*	$\delta_j^p = f_j'(\text{net}_j^p) \sum_n \delta_n^p w_{nj}$
	* where δ_n^p are from next layer (L^{k+1})
output derivative (assumes sigmoidal characteristic)	$f_j'(\text{net}_j^p) = o_j^p(1 - o_j^p)$

In the Hopfield network, every neuron is allowed to be connected to all other neurons, although the value of w_{ij} varies (it may also be 0 to indicate no unit interconnection). To avoid false reinforcement of a neuron state, the constraint $w_{ii} = 0$ is also employed. The w_{ij} values, therefore, play a fundamental role in the structure of the network. In general, a Hopfield network has significant interconnection (i.e., practical networks seldom have sparse W matrices, where $W = [w_{ij}]$).

Network Dynamics, Unit Firing Characteristic and State Propagation. A simple form for Hopfield neuron firing characteristics is the nonlinear threshold device:

$$o_i = \begin{cases} 1 & \text{if } \sum_{j;j\neq i} w_{ij} o_j > \alpha_i \\ 0 & \text{otherwise} \end{cases}$$

(Hopfield suggested an alternative characteristic, which leaves the output unchanged if $\Sigma_{j;j\neq i} w_{ij} o_j = \alpha_i$). Note in either case the neuron activation characteristic is nonlinear. Commonly, the threshold $\alpha_i = 0$. Viewing the state of a d-neuron Hopfield network at time (or iteration) t_k as an $d \times 1$ vector, $\boldsymbol{o}(t_k)$, the state of the system at time t_{k+1} (or iteration $k + 1$ in the discrete case) may be described by the nonlinear state transformation:

$$W\boldsymbol{o}(t_k) \overset{*}{\Rightarrow} \boldsymbol{o}(t_{k+1})$$

where the $\overset{*}{\Rightarrow}$ operator indicates the element by element state transition characteristic used to form $\boldsymbol{o}(t_{k+1})$. The model may be generalized for each unit to accomodate an additional vector of unit bias inputs.

The network state propagation suggests that the unit transitions are synchronous, that is, each unit, in lockstep fashion with all other units, computes its net activation and subsequent output. While this is achievable in (serial) simulations, it is not necessary. Also, empirical results have shown that it is not even necessary to update all units at each iteration. Surprisingly, network convergence is relatively insensitive to the fraction of units (15–100%) updated at each step.

Hopfield Energy Function and Storage Prescription. For the case of $\alpha_i = 0$, stable (stored) states correspond to minima of the following energy function:

$$E = -\left(\frac{1}{2}\right) \sum \sum_{i \neq j} w_{ij} o_i o_j$$

This leads to the rule for determination of w_{ij} and a set of desired stable states $\boldsymbol{o}^s$, $s = 1, 2, \ldots n$, that is the training set (stored states) $H = \{\boldsymbol{o}^1, \boldsymbol{o}^2, \ldots, \boldsymbol{o}^n\}$, as:

$$w_{ij} = \sum_{s=1}^{n} (2o_i^s - 1)(2o_j^s - 1) \qquad i \neq j$$

(with the previous constraint $w_{ii} = 0$). The convergence of the network to a stable state involves the Hamming distance between the initial state and the desired stable state. Different stable states which are close in Hamming distance are undesirable, since convergence to an incorrect stable state may result. Reference 42 suggests that an n-neuron network allows approximately $0.15n$ stable states; other researchers have proposed more conservative bounds (44).

Hopfield Pattern Recognition Example: Character Recall. Figure 12 shows a Hopfield network used as associative memory for recall of character data. A 10 × 10 pixel array is used to represent the character, yielding 100 pixels. Each pixel value is the state of a single, totally interconnected unit in a Hopfield network. Thus, the network consists of 100 units and approximately 100 × 100 interconnection weights. The network was trained using characters A, C, E and P. The top

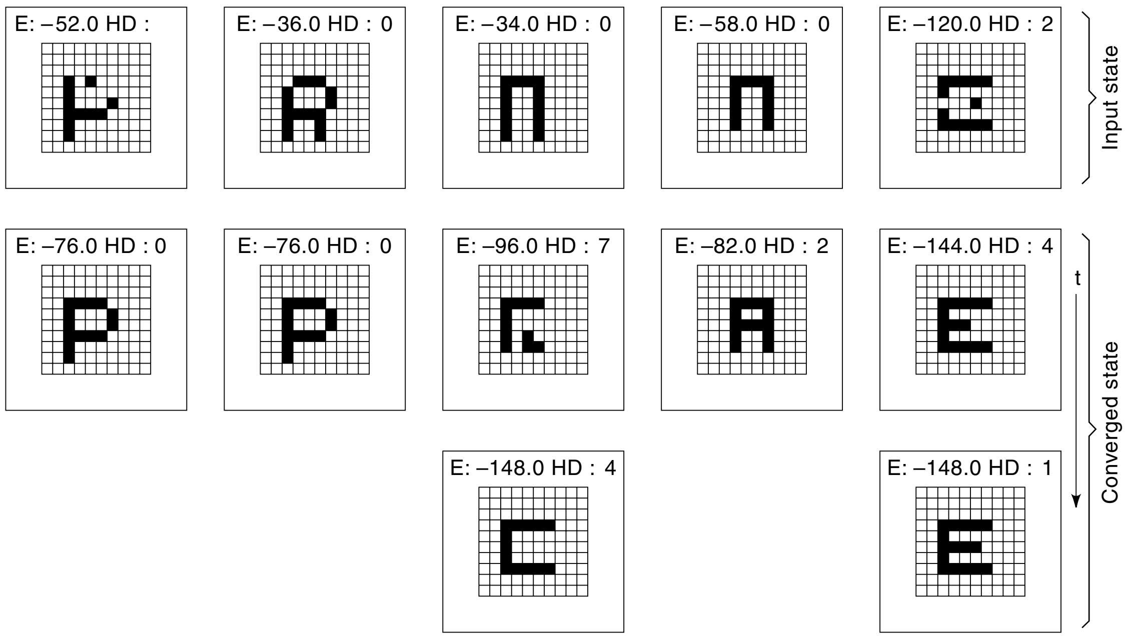

Figure 12. Use of a Hopfield network for character association/completion/recognition [from (1) Copyright 1989. Reprinted by permission of John Wiley & Sons.]

row of Fig. 12 shows initial states for the network; these are distorted patterns corresponding to the training patterns. Succeeding rows show the state evolution of the network. Note that the network converged to elements of H in at most two iterations in this example.

Kohonen Self-Organizing Feature Maps (SOFMs)

Kohonen (45,46) has shown an alternative neural learning structure involving networks which perform dimensionality reduction through conversion of feature space to yield topologically ordered similarity graphs or maps or clustering diagrams (with potential statistical interpretations). In addition, a lateral unit interaction function is used to implement a form of local competitive learning.

One-D and 2-D spatial configurations of units are used to form feature or pattern dimensionality reducing maps. For example, a 2-D topology yields a planar map, indexed by a 2-D coordinate system. of course, 3-D and higher dimensional maps are possible. Notice each unit, regardless of the topology, receives the input pattern $\boldsymbol{x} = (x_1, x_2 \; . \; . \; . \; x_d)^T$ in parallel. Considering the topological arrangement of the chosen units, the d-dimensional feature space is mapped into 1-D, 2-D, 3-D, and so on. The coordinate axes used to index the unit topology, however, have no explicit meaning or relation to feature space. They may, however, reflect a similarity relationship between units in the reduced dimensional space, where topological distance is proportional to dissimilarity.

Choosing the dimension of the feature map involves engineering judgement. Some PR applications naturally lead to a certain dimension, for example a 2-D map may be developed for speech recognition applications, where 2-D unit clusters represent phonemes (47). The dimensions of the chosen topological map may also influence the training time of the network. Once a topological dimension is chosen, the concept of a network neighborhood, (or cell or bubble) around each neuron may be introduced. The neighborhood, denoted N_c, is centered at neuron u_c, and the cell or neighborhood size (characterized by its radius in 2-D, for example) may vary with time (typically in the training phase). For example, initially N_c may start as the entire 2-D network, and the radius of N_c shrinks as iteration (described subsequently) proceeds. As a practical matter, the discrete nature of the 2-D net allows the neighborhood of a neuron to be defined in terms of nearest neighbors, for example, with a square array the four nearest neighbors of u_c are its N, S, E, and W neighbors; the eight nearest neighbors would include the corners.

Training the Self-Organizing Feature Maps. Each unit, u_i, in the network has the same number of weights as the dimension of the input vector, and receives the input pattern $\boldsymbol{x} = (x_1, x_2, \; . \; . \; . \; x_d)^T$ in parallel. The goal of the self-organizing network, given a large, unlabeled training set, is to have individual neural clusters self-organize to reflect input pattern similarity. Defining a weight vector for neural unit u_i as $\boldsymbol{m}_i = (w_{i1}, w_{i2}, \; . \; . \; . \; w_{id})^T$, the overall structure may be viewed as an array of matched filters, which competitively adjust unit input weights on the basis of the current weights and goodness of match. A useful viewpoint is that each unit tries to become a matched filter, in competition with other units.

Assume the network is initialized with the weights of all units chosen randomly. Thereafter, at each training iteration, denoted k for an input pattern $\boldsymbol{x}(k)$, a distance measure $d(\boldsymbol{x}, \boldsymbol{m}_i)$ between $\boldsymbol{x}$ and $\boldsymbol{m}_i$ $\forall i$ in the network is computed. This may be an inner product measure (correlation), Euclidean distance, or another suitable measure. For simplicity, we proceed using the Euclidean distance. For pattern $\boldsymbol{x}(k)$, a matching phase is used to define a winner unit u_c, with weight vector $\boldsymbol{m}_c$, using

$$\|\boldsymbol{x}(k) - \boldsymbol{m}_c(k)\| = \overset{\min}{i} \{\|\boldsymbol{x}(k) - \boldsymbol{m}_i(k)\|\}$$

Thus, at iteration k, given $\boldsymbol{x}$, c is the index of the best matching unit. This affects all units in the currently defined cell, bubble, or cluster surrounding u_c, $N_c(k)$ through the global network updating phase as follows:

$$\boldsymbol{m}_i(k+1) = \begin{cases} \boldsymbol{m}_i(k) + \alpha(k)[\boldsymbol{x}(k) - \boldsymbol{m}_i(k)] & i \in N_c \\ \boldsymbol{m}_i(k) & i \notin N_c \end{cases}$$

The updating strategy bears a strong similarity to the c-means algorithm. $d(\boldsymbol{x}, \boldsymbol{m}_i)$ is decreased for units inside N_c, by moving $\boldsymbol{m}_i$ in the direction $(\boldsymbol{x} - \boldsymbol{n}_i)$. Therefore, after the adjustment, the weight vectors in N_c are closer to input pattern $\boldsymbol{x}$. Weight vectors for units outside N_c are left uncharged. The competitive nature of the algorithm is evident since after the training iteration units outside N_c are *relatively* further from $\boldsymbol{x}$. That is, there is an opportunity cost of not being adjusted. Again, α is a possibly iteration-dependent design parameter.

The resulting accuracy of the mapping depends upon the choices of N_c, $\alpha(k)$ and the number of iterations. Kohonen cites the use of 10,000–100,000 iterations as typical. Furthermore, $\alpha(k)$ should start with a value close to 1.0, and gradually decrease with k. Similarly, the neighborhood size, $N_c(k)$, deserves careful consideration in algorithm design. Too small a

	Item																															
	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	1	2	3	4	5	6
Attribute																																
a1	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a2	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a3	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
a4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
a5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

(a)

B	C	D	E	*	Q	R	*	Y	Z
A	*	*	*	*	P	*	*	X	*
*	F	*	N	O	*	W	*	*	1
*	G	*	M	*	*	*	*	2	*
H	K	L	V	T	U	*	3	*	*
*	1	*	*	*	*	*	*	4	*
*	J	*	S	*	*	V	*	5	6

(b)

Figure 13. Sample results using a 2-D Kohonen SOFM for a 5-D feature case involving uppercase characters [from (48) Copyright 1988, IEEE]. Part (a) shows the extracted features for each character. Part (b) shows the resulting map.

choice of $N_c(0)$ may lead to maps without topological ordering. Therefore, it is reasonable to let $N_c(0)$ be fairly large (Kohonen suggests one-half the diameter of the map) shrinking $N_c(k)$ (perhaps linearly) with k to the fine-adjustment phase, where N_c only consists of the nearest neighbors of unit u_c. Of course, a limiting case is where $N_c(k)$ becomes one unit. Additional details of the self-organizing algorithm are summarized in the cited references.

Example: Self Organizing Feature Map Application to Unsupervised Learning. Figure 13, [from (48)], shows sample results for a 5-D feature vector case. Uppercase characters are presented as unlabeled training data to a 2-D SOFM. Figure 13(a) shows the unlabeled training set samples H_u; Fig. 13(b) shows the self-organized map resulting from the algorithm. As evidenced by Fig. 13(b), 2-D clustering of the different dimensionality-reduced input patterns occurs. As in other learning examples, vectors were chosen randomly from H_u at each iteration. $\alpha(k)$ decreased linearly with k from 0.5 ($= \alpha(o)$) to 0.04 for $k \leq 10{,}000$. Similarly, for this simulation the 2-D map was chosen to be of hexagonal structure with 7×10 units. For $k \leq 1000$, the radius of N_c decreased from 6 (almost all of the network) to 1 (u_c and its six nearest neighbors).

BIBLIOGRAPHY

1. R. J. Schalkoff, *Digital Image Processing,* New York: Wiley, 1989.
2. R. J. Schalkoff, *Pattern Recognition: Statistical, Syntactic and Neural Approaches,* New York: Wiley, 1992.
3. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* NY: Wiley, 1973.
4. P. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach,* Englewood Cliffs, NJ: Prentice-Hall, 1982.
5. K. Fukunaga, *Introduction to Statistical Pattern Recognition,* N.Y.: Academic Press, 1990.
6. S. T. Bow, *Pattern Recognition,* New York: Marcel Dekker, 1984.
7. S. Watanabe, *Pattern Recognition: Human and Mechanical,* New York: Wiley, 1985.
8. Y. T. Chien, *Interactive Pattern Recognition,* New York: Dekker, 1978.
9. E. A. Patrick, *Fundamentals of Pattern Recognition,* Englewood Cliffs, NJ: Prentice-Hall, 1972.
10. C. W. Therrien, *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics,* New York: Wiley, 1989.
11. R. A. Fisher, The use of multiple measurements in taxonomic problems, (reprinted in) *Contributions to Mathematical Statistics,* New York: Wiley, 1950.
12. Y. C. Ho and R. L. Kayshap, An algorithm for linear inequalities and its application, *IEEE Trans. Electron. Comput.,* **EC-14**: 683–688, 1965.
13. K. Fukunaga and D. R. Olsen, Piecewise linear discriminant functions and classification errors for multiclass problems, *IEEE Trans. Inf. Theory,* **IT-16**: 99–100, 1970.
14. A. K. Jain and R. Dubes, *Algorithms for Clustering Data,* Englewood Cliffs, NJ: Prentice-Hall, 1988.
15. G. B. Coleman and H. C. Andrews, Image segmentation by clustering, *Proc. IEEE,* **67**: 773–785, 1979.
16. J. Bryant, On the clustering of multidimensional pictorial data, *Pattern Recognition,* **11**: 115–125, 1979.
17. R. K. Blashfield, M. S. Aldenderfer, and L. C. Morey, Cluster analysis software, in *Handbook of Statistics,* Vol. 2, P. R. Krishniah and L. N. Kanal, eds., Amsterdam, The Netherlands: North Holland, 245–266, 1982.
18. R. C. Dubes and A. K. Jain, Clustering techniques: The user's dilemma, *Pattern Recognition,* **8**: 247–260, 1976.
19. K. S. Fu, *Syntactic Pattern Recognition and Applications,* Englewood Cliffs, NJ: Prentice-Hall, 1982.
20. J. Tou and R. C. Gonzalez, *Pattern Recognition Principles,* Reading, MA: Addison-Wesley, 1974.
21. R. C. Gonzalez and M. G. Thomason, *Syntactic Pattern Recognition,* Reading, MA: Addison-Wesley, 1978.
22. L. Miclet, *Structural Methods in Pattern Recognition,* New York: Springer-Verlag, 1986.
23. T. Pavlidis, *Structural Pattern Recognition,* New York: Springer-Verlag, 1977.
24. K. S. Fu, A Step Towards Unification of Syntactic and Statistical Pattern Recognition, *IEEE Trans. Pattern Anal. Mach. Intell.,* **PAMI-8**: 398–404, 1986.
25. H. S. Don and K. S. Fu, A syntactic method for image segmentation and object recognition, *Pattern Recognition,* **18** (1): 73–87, 1985.
26. J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata,* Reading, MA: Addison-Wesley, 1969.
27. R. N. Moll, M. A. Arbib, and A. J. Kfoury (eds.) *An Introduction to formal Language Theory,* New York: Springer-Verlag, 1988.
28. H. C. Lee and K. S. Fu, A stochastic syntactic analysis procedure and its application to pattern classification, *IEEE Trans. Comput.,* **C-21**: 660–666, 1972.
29. A. Sanfeliu and K. S. Fu, A distance measure between attributed relational graphs for pattern recognition, *IEEE Trans. Syst. Man. Cybern.,* **SMC-13**: 353–362, 1983.
30. L. G. Shapiro and R. M. Haralick, A metric for comparing relational descriptions, *IEEE Trans. Pattern Anal. Mach. Intell.,* **7**: 90–94, 1985.
31. R. J. Schalkoff, *Artificial Neural Networks,* New York: McGraw-Hill, 1997.
32. J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research,* Cambridge, MA: MIT Press, 1988.
33. D. E. Rummelhart and J. L. McClelland, *Parallel Distributed Processing—Explorations in the Microstructure of Cognition, Volume 1: Foundations,* Cambridge, MA: MIT Press, 1986.
34. D. E. Rummelhart and J. L. McClelland, *Parallel Distributed Processing—Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models,* Cambridge, MA: MIT Press, 1986.
35. R. Rosenblatt, *Principles of Neurodynamics,* New York: Spartan Books, 1959.
36. M. Minsky and S. Papert, *Perceptrons—An Introduction to Computational Geometry,* Cambridge, MA: MIT Press, 1969.
37. N. J. Nilsson, *Learning Machines,* New York: McGraw-Hill, 1965. (This has been revised as *Mathematical Foundations of Learning Machines,* Morgan-Kaufmann, San Mateo, CA, 1989.)
38. B. Widrow and M. E. Hoff, Adaptive switching circuits, *1960 IRE WESCON Conv. Record,* Part 4, 96–104 Aug. 1960, (reprinted in [Anderson/Rosenfeld 1988]).
39. W. Y. Huang and R. P. Lippmann, Comparison between neural net and conventional classifiers, *Proc. IEEE Int. Conf. Neural Netw.,* San Diego, **IV**: 485–493, June 1987.
40. B. Widrow and M. A. Lehr, 30 years of adaptive neural networks: perceptron, madaline and backpropagation, *Proc. IEEE,* **78**: 1415–1442, 1990.
41. B. Widrow and R. G. Winter, Neural nets for adaptive filtering and adaptive pattern recognition, *IEEE Computer,* **21:25**: 39, March 1988.

42. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci.,* **79** (Biophysics), 2554–2558, April 1982.

43. J. J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci.,* **81** (Biophysics), 3088–3092, May 1984.

44. Y. S. Abu-Mostafa and J. M. St. Jacques, Information Capacity of the Hopfield Model, *IEEE Trans. Inf. Theory,* **IT-31**: 461–464, 1985.

45. T. Kohonen, *Self-Organization and Associative Memory,* Berlin: Springer-Verlag, 1984.

46. J. A. Kangas, T. Kohonen, and J. T. Laaksonen, Variants of Self-Organizing Maps, *IEEE Trans. on Neural Netw.,* **1**: 93–99, 1990.

47. B. D. Shriver, Artificial neural systems, *IEEE Computer,* **21**, 3: March 1988.

48. T. Kohonen, Self-Organizing Feature Maps, tutorial course notes from 1988 Conference on Neural Networks, San Diego, CA., 1988. (Accompanying videotape available from the Institute of Electrical and Electronics Engineers, Inc., 345 E. 47th St., N.Y., N.Y., 10017.)

Reading List

Reference Journals. Work on various aspects of PR continues to cross-pollinate journals. Useful sources include: *Pattern Recognition Letters, Pattern Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Systems, Man and Cybernetics, IEEE Transactions on Geoscience and Remote Sensing, IEEE Transactions on Neural Networks,* and *Image and Vision Computing.*

ROBERT J. SCHALKOFF
Clemson University