# 6 Unsupervised Learning in Multilayer Neural Networks

The task of pattern clustering is to automatically group unlabeled input vectors into several categories (clusters) so that each input is assigned a label corresponding to a unique cluster. The clustering process is normally driven by a similarity measure. Vectors in the same cluster are similar, which usually means that they are "close" to each other in the input space. A simple clustering net that employs competitive learning was covered in Section 4.3. In this section, two additional clustering neural networks are described which have more interesting features than the simple competitive net of Chapter 4. Either network is capable of automatic discovery (estimation) of the underlying number of clusters.

There are various ways to represent clusters. The first network described uses a simple representation in which each cluster is represented by the weight vector of a prototype unit (this is also the prototype representation scheme adapted by the simple competitive net). The first network is also characterized by its ability to allocate clusters incrementally. Networks with incremental clustering capability can handle an infinite stream of input data. They do not require large memory to store training data because their cluster prototype units contain implicit representation of all the inputs previously encountered.

The second clustering network described in this section has a more complex architecture than the first net. It employs a distributed representation as opposed to a single-prototype-unit cluster representation. This network is nonincremental in terms of cluster formation. However, the highly nonlinear multiple-layer architecture of this clustering net enables it to perform well on very difficult clustering tasks. Another interesting property of this net is that it does not require an explicit user-defined similarity measure. The network develops its own internal measure of similarity as part of the training phase.

## *6.1 Adaptive Resonance Theory (ART) Networks*

Adaptive resonance architectures are artificial neural networks that are capable of stable categorization of an arbitrary sequence of unlabeled input patterns in real time. These architectures are capable of continuous training with nonstationary inputs. They also solve the *stability-plasticity dilemma*; namely, they let the network adapt yet prevent current inputs from destroying past training. The basic principles of the underlying theory of these networks, known as adaptive *resonance theory* (ART), were introduced by Grossberg (1976). ART networks are biologically motivated and were developed as possible models of cognitive phenomena in humans and animals.

A class of ART architectures, called ART1, is characterized by a system of ordinary differential equations (Carpenter and Grossberg, 1987a), with associated theorems proving its self-stabilization property and the convergence of its adaptive weights. ART1 embodies certain simplifying assumptions that allow its behavior to be described in terms of a discrete-time clustering algorithm. A number of interpretations/simplifications of the ART1 net have been reported in the literature (Lippmann, 1987; Moore, 1989). In the following, Moore's abstraction of the clustering algorithm from the ART1 architecture is adopted, and this algorithm is

discussed in conjunction with a simplified architecture.

The basic architecture of the ART1 net consists of a layer of linear units representing prototype vectors whose outputs are acted on by a winner-take-all network (described in Section 4.3). This architecture is identical to that of the simple competitive net in Figure 4-6 with one major difference: The linear prototype units are allocated dynamically, as needed, in response to novel input vectors. Once a prototype unit is allocated, appropriate lateral-inhibitory and self-excitatory connections are introduced so that the allocated unit may compete with preexisting prototype units. Alternatively, one may assume a prewired architecture as in Figure 4-6 with a large number of inactive (zero weight) units. Here, a unit becomes active if the training algorithm decides to assign it as a cluster prototype unit, and its weights are adapted accordingly.

The general idea behind ART1 training is as follows. Every training iteration consists of taking a training example $\mathbf{x}^k$ and examining existing prototypes (weight vectors $\mathbf{w}_j$) that are sufficiently similar to $\mathbf{x}^k$. If a prototype $\mathbf{w}_i$ is found to "match" $\mathbf{x}^k$ (according to a "similarity" test based on a preset matching threshold), example $\mathbf{x}^k$ is added to the cluster represented by $\mathbf{w}_i$, and $\mathbf{w}_i$ is modified to make it better match $\mathbf{x}^k$ If no prototype matches $\mathbf{x}^k$, then $\mathbf{x}^k$ becomes the prototype for a new cluster. The details of the ART1 clustering procedure are considered next.

The input vector (pattern) **x** in ART1 is restricted to binary values, $\mathbf{x} \in \{0,1\}^n$. Each learned cluster, say, cluster *j*, is represented by the weight vector $\mathbf{w}_j \in \{0,1\}^n$ of the *j*th prototype unit. Every time an input vector **x** is presented to the ART1 net, each existing prototype unit computes a normalized output (the motivation behind this normalization is discussed later)

$$y_j = \frac{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}}{\left\|\mathbf{w}_j\right\|^2} \tag{6.1}$$

and feeds it to the winner-take-all net for determining the winner unit[1]. Note that $y_j$ is the ratio of the overlap between prototype $\mathbf{w}_j$ and **x** to the size of $\mathbf{w}_j$. The winner-take-all net computes a "winner" unit *i*. Subject to further verification, the weight vector of the winner unit $\mathbf{w}_i$ now represents a potential prototype for the input vector. The verification comes in the form of passing two tests.

In order to pass the first test, the input **x** must be "close enough" to the winner prototype $\mathbf{w}_i$; i.e.,

$$y_i = \frac{\mathbf{w}_i^{\mathrm{T}}\mathbf{x}}{\left\|\mathbf{w}_i\right\|^2} > \frac{\left\|\mathbf{x}\right\|^2}{n} \tag{6.2}$$

[1] In practice, a small positive constant, say, 0.5, is added to the denominator in Equation (6.1) in order to avoid division by zero.

Here, $\|\mathbf{x}\|^2$ and $\|w_i\|^2$ are equal to the number of 1s in **x** and $w_i$, respectively. Passing this test guarantees that a sufficient fraction of the $\mathbf{w}_i$ and **x** bits is matched.

The second test is a match verification test between $\mathbf{w}_i$ and **x**. This test is passed if

$$\frac{\mathbf{w}_i^{\mathrm{T}}\mathbf{x}}{\|\mathbf{x}\|^2} \geq \rho \tag{6.3}$$

where $0 < \rho < 1$ is a user-defined vigilance parameter. Here, $\mathbf{w}_i$ is declared to "match" **x** if a significant fraction (determined by $\rho$) of the 1s in **x** appears in $\mathbf{w}_i$. Note that Equation (6.3) causes more differentiation among input vectors of smaller magnitude. This feature of ART1 is referred to as *automatic scaling, self-scaling, or noise insensitivity*.

If these two tests are passed by the winner unit *i* for a given input **x** (here, the network is said to be in *resonance*), then **x** joins cluster *i*, and this unit's weight vector $\mathbf{w}_i$ is updated according to

$$\mathbf{w}_i^{\mathbf{new}} = \mathbf{w}_i \wedge \mathbf{x} \tag{6.4}$$

where "$\wedge$" stands for the logical AND operation applied component-wise to the corresponding components of vectors $\mathbf{w}_i$ and $\mathbf{x}_i$. According to Equation (6.4), a new prototype $\mathbf{w}_i$ can only have fewer and fewer 1s as training progresses. Note that it is possible for a training example to join a new cluster but eventually to leave that cluster because other training examples have joined it.

The second scenario corresponds to unit *i* passing the first test but not the second one. Here, the *i*th unit is deactivated (its output is clamped to zero until a new input arrives) and the tests are repeated with the unit with the next highest normalized output. If this scenario persists even after all existing prototype units are exhausted, then a new unit representing a new cluster *j* is allocated and its weight vector is initialized according to $\mathbf{w}_j$

$$\mathbf{w}_j = \mathbf{x} \tag{6.5}$$

In a third scenario, unit *i* does not pass the first test. Here, $\mathbf{w}_i$ is declared "too far" from the input **x**, and a new unit representing a new cluster, *j*, is allocated with its weight vector $\mathbf{w}_j$ initialized as in Equation (6.5). Hence, initially, $\mathbf{w}_j$ is a binary vector. And since, $\mathbf{w}_j$ is updated according to Equation (6.4), the vector $\mathbf{w}_j$ preserves its binary nature. This is true for any unit in the ART1 net, since all units undergo the initialization in Equation (6.5) upon being allocated.

Note that the learning dynamics in the second scenario described above constitute a search through the prototype vectors, looking at the closest, next closest, etc., according to the criterion $\mathrm{i}\left(\mathbf{w}_j^{\mathrm{T}}\mathbf{x}\right)/\|\mathbf{w}_j\|^2$. This search is continued until a prototype vector is found that satisfies the matching criteria in Equation (6.3). These criteria

are different. The first criterion measures the fraction of the bits in $\mathbf{w}_j$ that are also in **x**, whereas the second criterion measures the fraction of the bits in **x** that are also in $\mathbf{w}_j$.

Therefore, going further away by the first measure may actually bring us closer by the second. It also should be noted that this search only occurs before stability is reached for a given training set. After that, each prototype vector is matched on the first attempt, and no search is needed.

The normalization factor $\|\mathbf{w}_j\|^2$ in Equation (6.1) is used as a "tie breaker." It favors smaller-magnitude prototype vectors over vectors that are supersets of them (i.e., have 1s in the same places) when an input matches them equally well[2]. This mechanism of favoring small prototype vectors helps maintain prototype vectors apart. It also helps compensate for the fact that in the updating step of Equation (6.4), the prototype vectors always move to vectors with fewer 1s.

The vigilance parameter $\rho$ in Equation (6.3) controls the granularity of the clusters generated by the ART1 net. Small $\rho$ values allow for large deviations from cluster centers and hence lead to a small set of clusters. On the other hand, a higher vigilance leads to a larger number of tight clusters. Regardless of the setting of $\rho$, the ART1 network is stable for a finite training set; i.e., the final clusters will not change if additional training is performed with one or more patterns drawn from the original training set. A key feature of the ART1 network is its continuous learning ability. This feature, coupled with the preceding stability result, allows the ART1 net to follow nonstationary input distributions.

The clustering behavior of the ART1 network is illustrated for a set of random binary vectors. Here, the task of ART1 is to cluster 24 uniformly distributed random vectors $\mathbf{x} \in \{0,1\}^{16}$. Simulation results are shown in Figure 6-1(a) and (b) for $\rho = 0.5$ and $\rho = 0.7$, respectively (here, the vectors are shown as 4 x 4 patterns of "on" and "off" pixels for ease of visualization). The resulting prototype vectors are also shown. Note the effect of the vigilance parameter setting on the granularity of the generated clusters.

The family of ART networks also includes more complex models such as ART2 (Carpenter and Grossberg, 1987b) and ART3 (Carpenter and Grossberg, 1990). These ART models are capable of clustering binary and analog input patterns. However, these models are inefficient from a computational point of view. A simplified model of ART2, ART2-A, has been proposed that is two to three orders of magnitude faster than ART2. Also, a supervised real-time learning ART model

---

[2] For example, the vector $\mathbf{w}_1 = [1\ 0\ 1\ 0]^T$ is favored over its superset $\mathbf{w}_1 = [1\ 1\ 1\ 0]^T$ when matching the example $\mathbf{x} = [1\ 0\ 1\ 1]^T$ Here, $\mathbf{w}_1^T\mathbf{x} = \mathbf{w}_2^T\mathbf{x} = 2$ while

$$\frac{\mathbf{w}_1^T\mathbf{x}}{\|\mathbf{w}_1\|^2} = 1 > \frac{\mathbf{w}_2^T\mathbf{x}}{\|\mathbf{w}_2\|^2} = f.$$

called ARTMAP has been proposed.

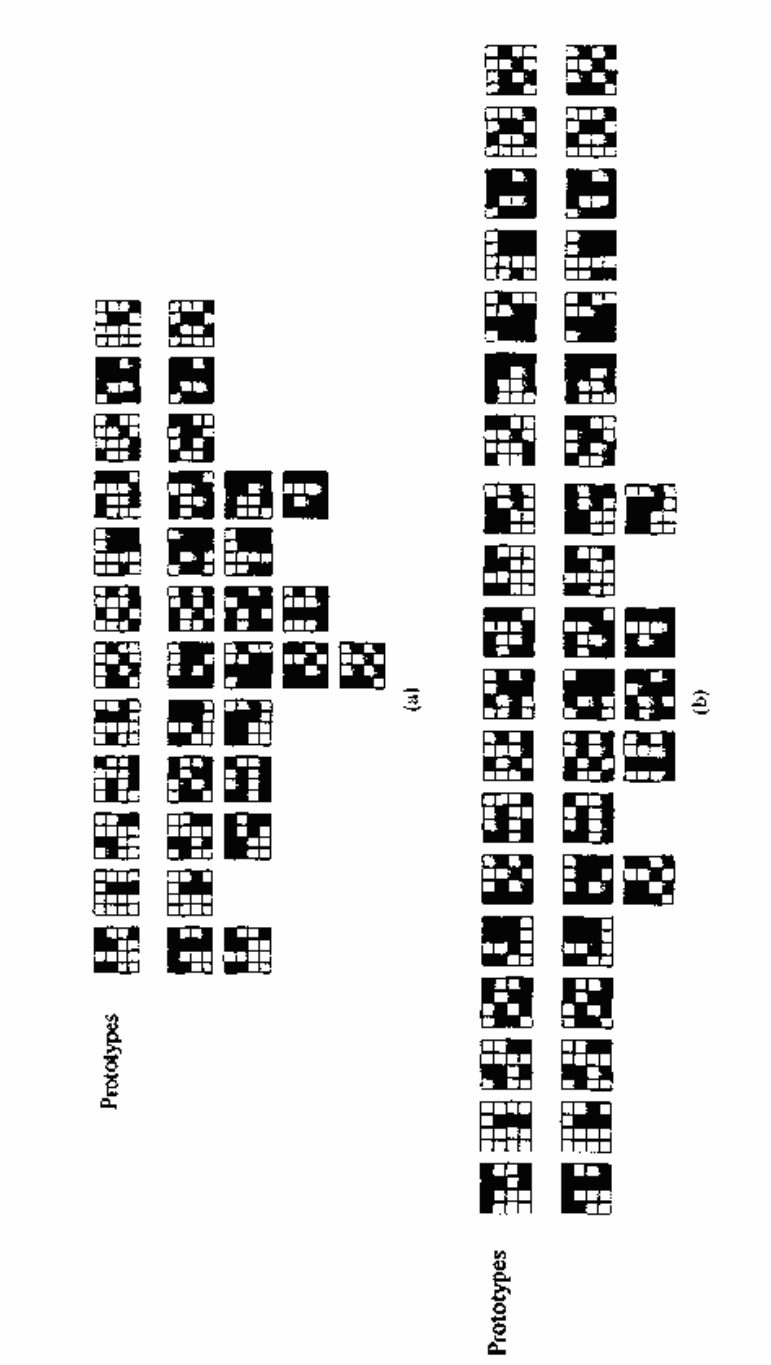


**Figure 6-1** Random binary pattern clustering emplying the Art1 net. Different vigilance values causes different numbers of categories (clusters) to form (*a*) $\rho = 0.5$ and (*b*) $\rho = 0.7$. For each case, the top row shows prototype vectors extracted by the ART1 network. [From Hassoun, 1995]

An example of ART2 clustering is shown in Figure 6-2 (Carpenter and Grossberg, 1987b). Here, the problem is to cluster a set of fifty 25-dimensional real-valued input signals (patterns). The results are shown for two different vigilance levels. It is left to the reader to check (subjectively) the consistency of the formed clusters. Characterization of the clustering behavior of ART2 was given by Burke (1991), who draws an analogy between ART-based clustering and fc-means-based clustering.

ART networks are sensitive to the presentation order of the input patterns; they may yield different clustering on the same data when the presentation order of patterns is varied (with all other parameters kept fixed). Similar effects are also present in incremental versions of classical clustering methods such as *k*-means clustering (i.e., *k*-means is also sensitive to the initial choice of cluster centers).

## *6.2 Autoassociative Clustering Network*

Other data-clustering networks can be derived from *concept-forming cognitive models* (Anderson and Murphy, 1986). A *concept* describes the situation where a number of different objects are categorized together by some rule or similarity relationship. For example, a person is able to recognize that physically different objects are really "the same" (e.g., a person's concept of "tree").

A simple concept-forming model consists of two basic interrelated components. First, it consists of a prototype-forming component, which is responsible for generating category prototypes via an autoassociative learning mechanism. The second component is a retrieval mechanism where a prototype becomes an attractor in a dynamical system. Here, the prototype and its surrounding basin of attraction represent an individual concept.

Artificial neural networks based on the preceding concept-forming model have been proposed for data clustering of noisy, superimposed patterns (Hassoun et al., 1992). Here, a feedforward (single- or multiple-layer) net is trained in an autoassociative mode with the noisy patterns. The training is supervised in the sense that each input pattern to the net serves as its own target. However, these targets are not the "correct answers" in the general sense of supervised training. The strategy here is to force the network to develop internal representations during training so as to better reconstruct the noisy inputs. In the prototype-extraction phase, the trained feedforward net is transformed into a dynamical system by using the output-layer outputs as inputs to the net, thus forming an external feedback loop. Now, and with proper stabilization, when initialized with one of the input patterns, the dynamical system will evolve and eventually converge to the "closest" attractor state. Hopefully, these attractors may be identified with the prototypes derived from the training phase.

Next, these ideas are described in the context of a simple single-layer network, and then a more general auto-associative clustering architecture is presented. Consider an unlabeled training set of vectors $\hat{\mathbf{x}} \in R^n$ representing distorted and/or noisy versions of a set $\left\{\mathbf{x}^k : k = 1, 2, ..., m\right\}$ of $m$ unknown prototype vectors. Also assume a

single-layer net of $n$ linear units each having an associated weight vector $\mathbf{w}_i, i = 1, 2, ..., n.$ Let us denote by $\mathbf{W}$ the $n \times n$ matrix whose ith row is the weight vector $\mathbf{w}_i^{\mathrm{T}}$. This simple network outputs the vector $y \in R^n$ upon the presentation of an input $\hat{\mathbf{x}}$ where $\mathbf{y} = \mathbf{W}\hat{\mathbf{x}}$. Now the connection matrix $\mathbf{W}$ is updated in response to the presentation of a sample $\hat{\mathbf{x}}^k$ according to the matrix form of the Widrow-Hoff (LMS) rule

$$\Delta\mathbf{W} = \rho\left(\hat{\mathbf{x}}^k - y^k\right)\hat{\mathbf{x}}^{k^{\mathrm{T}}} \tag{6.6}$$

which realizes a gradient-descent minimization of the criterion function

$$J\left(\mathbf{W}\right) = \frac{1}{2}\sum_k \left\|\hat{\mathbf{x}}^k - \mathbf{W}\hat{\mathbf{x}}^k\right\|^2 \tag{6.7}$$

Therefore, the trained net will be represented by a connection matrix $\mathbf{W}$ that approximates the mapping

$$\mathbf{W}\mathbf{x} = \mathbf{x} \tag{6.8}$$

for all $m$ prototype vectors $\mathbf{x}$. This approximation results from minimizing $J$ in Equation (6.7), and it requires that the clusters of noisy samples $\hat{\mathbf{x}}$ associated with each prototype vector be sufficiently uncorrelated. In addition, it requires that the network be incapable of memorizing the training autoassociations and that the number of underlying prototypes $m$ to be learned be much smaller than $n$.

According to Equation (6.8), the autoassociative training phase attempts to estimate the unknown prototypes and encode them as eigenvectors of $\mathbf{W}$ with eigenvalues of 1. A simple method for extracting these eigenvalues (prototypes) is to use the "power" method of eigenvector extraction. This is an iterative method that can be used to pick out the eigenvector with the largest-magnitude eigenvalue of a matrix $\mathbf{A}$ by repeatedly passing an initially random vector $\mathbf{c}^0$ through the matrix according to

$$\mathbf{c}^{k+1} = \mathbf{A}\mathbf{c}^k \qquad k = 0, 1, 2, .... \tag{6.9}$$

After a number of iterations, the eigenvector with the largest-magnitude eigenvalue will dominate. This eigenvector extraction method can be readily implemented by adding external feedback to our simple net, with $\mathbf{A}$ represented by $\mathbf{W}$ and $\mathbf{c}$ by $\mathbf{y}$ or $\mathbf{x}$.

Once initialized with one of the noisy vectors $\hat{\mathbf{x}}$ the resulting dynamical system evolves its state (the $n$-dimensional output vector $\mathbf{y}$) in such a way that this state moves toward the prototype vector $\mathbf{x}$ "most similar" to $\hat{\mathbf{x}}$. This is possible because the prototype vectors $\mathbf{x}$ approximate the dominating eigenvectors of $\mathbf{W}$ (with eigenvalues close to 1). Note that the remaining eigenvectors of $\mathbf{W}$ arise from learning uncorrelated noise and tend to have small eigenvalues compared with 1. The ability of the auto-associative dynamical net to selectively extract a learned prototype/eigenvector from a "similar" input vector, as opposed to always extracting the most dominant eigenvector, is due to the fact that all learned prototypes have comparable eigenvalues close to unity. Thus the extracted prototype is the one that is "closest" to the initial state (input vector) of the net.

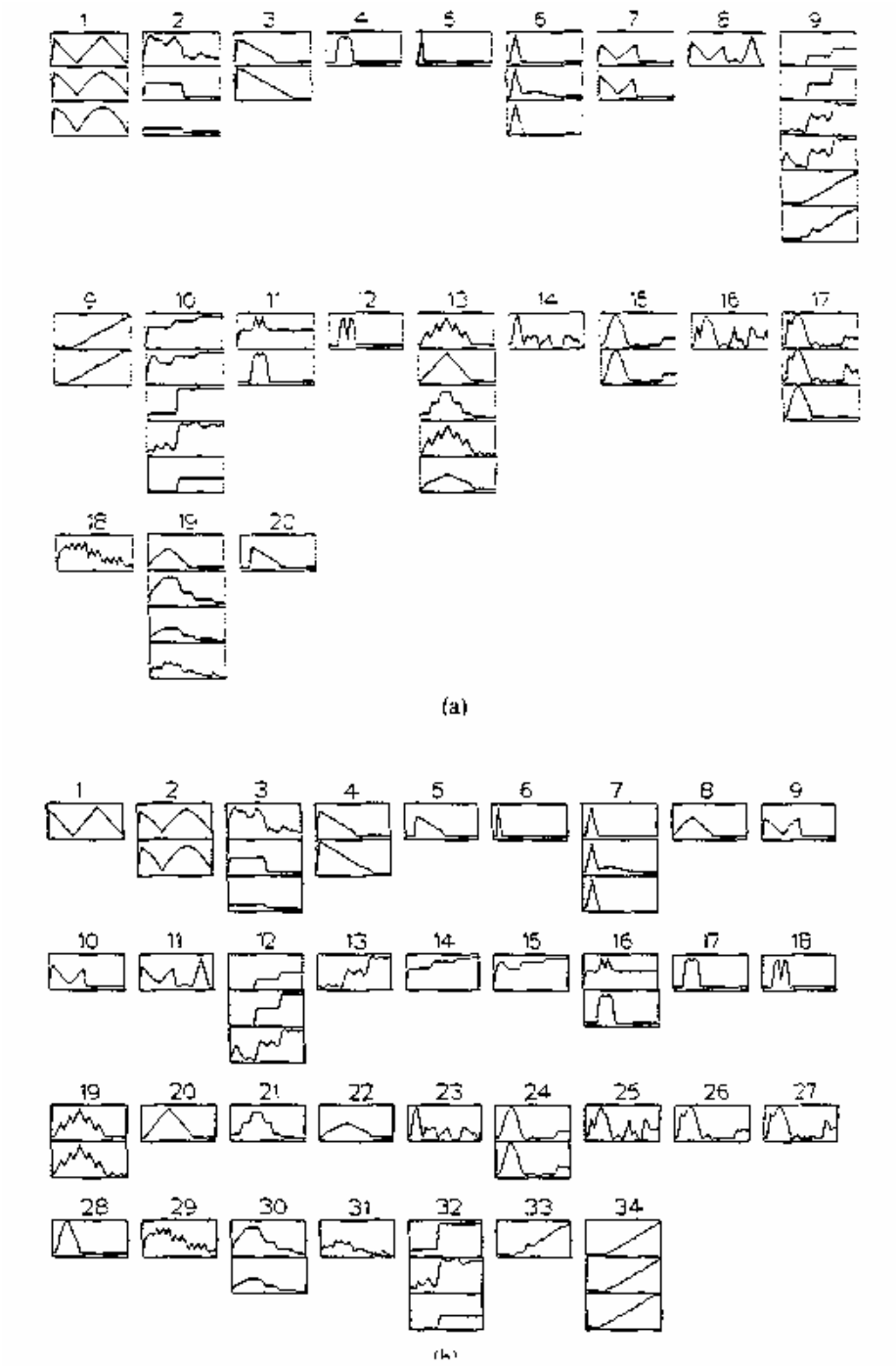

**Figure 6-2** ART2 clustering of analog signals for two vigilance levels. The vigilance value in part (a) is smaller than that in part (b). Note that cluster number 9 has eight signals and is broken down into two separate columns for display reasons onlv [From Carpenter and Grossberg, 1987b; © 1987 Optical Society of America )

The stability of the dynamic autoassociative net is an important design issue. Stability is determined by the network weights and network architecture. Care must be taken in matching the learning algorithm for prototype encoding in the feedforward net to the dynamic architecture that is ultimately used for prototype extraction. One would like to design an autoassociative clustering net that minimizes an associated energy or Liapunov function; i.e., starting from any initial state, the system's state always evolves along a trajectory for which the energy function is monotonically decreasing. Anderson et al. (1990) reported a stable autoassociative clustering net based on the brain-state-in-a-box (BSB) concept-forming model.

A serious limitation of the autoassociative linear net just described is that it does not allow the user to control the granularity of the formed clusters; i.e., the number of learned prototypes. This network will tend to merge different clusters that are "close" to each other in the input space due to the lack of cluster competition mechanisms (recall the similarity and vigilance tests employed in the ART1 net for controlling cluster granularity). When two different clusters are merged by the linear net, they become represented by a distorted prototype that is a linear combination of the two correct (but unknown) prototypes. Introducing nonlinearity into the autoassociative net architecture can help the net overcome this limitation by allowing control of the granularity of the clusters. This feature is discussed in connection with the dynamic nonlinear multilayer autoassociative network (Hassoun et al., 1992) considered next.

Consider a two-hidden-layer feedforward net with an output layer of linear units. All hidden-layer units employ the hyperbolic tangent activation function

$$f(\xi) = \tanh(\beta\xi) \tag{6.10}$$

where $\beta$ controls the slope of the activation. The activation slopes of the second-hidden-layer units (the layer between the first hidden and output layers) are fixed (typically set to 1). On the other hand, the activation slopes for the units in the first hidden layer are made monotonically increasing during training, as explained below. Each hidden unit in the first layer receives inputs from all components of an $n$-dimensional input vector and an additional bias input (held fixed at 1). Similarly, each unit in the second hidden layer receives inputs from all units in the first hidden layer plus a bias input of 1. Finally, each linear output unit receives inputs from all second-hidden-layer units plus a bias.

This layered network functions as an autoassociative net, as described above. Therefore, the $n$-output-layer units serve to reconstruct (decode) the $n$-dimensional vector presented at the input. The network is to discover a limited number of representations (prototypes) of a set of noisy input vectors to describe the training data. An essential feature of the network's architecture is therefore a restrictive "bottleneck" in the hidden layer; the number of units in each hidden layer (especially the first hidden layer) is small compared to $n$. The effect of this bottleneck is to restrict the degrees of freedom of the network and constrain it to discovering a limited set of unique prototypes that describes (clusters) the training set. The

network does not have sufficient capacity to memorize the training set. This clustering is further enhanced by aspects of the learning algorithm, described below. Autoassociative multilayer nets with hidden-layer bottleneck have been studied by Bouriard and Kamp (1988), Baldi and Hornik (1989), Funahashi (1990), Kramer (1991), among others. In these studies, such nets have been found to implement principal-component analysis (PCA) and nonlinear PCA when one hidden layer and three or more hidden layers are used, respectively.

The learning algorithm employed is essentially the incremental backprop algorithm (Enachescu, 2003) with simple heuristic modifications. These heuristics significantly enhance the network's tendency to discover the best prototypical representations of the training data. These modifications include a dynamic slope for the first-hidden-layer activations that saturates during learning and damped learning-rate coefficients. As a result of learning, the first hidden layer in this net discovers a set of bipolar binary distributed representations that characterize the various input vectors. The second hidden and the output layers perform a nonlinear mapping that decodes these representations into reduced-noise versions of the input vectors.

In order to enhance separation of clusters and promote grouping of similar input vectors, the slope $\beta$ of the activation functions of first-hidden-layer units is made dynamic; it increases monotonically during learning, according to

$$\beta(k) = i^{-k} \tag{6.11}$$

where $i^{-k}$ is greater than but close to 1 and $k$ is the learning-step index. As a result, the nonlinearity gradually (over a period of many cycles through the whole training set) becomes the sgn (sign) function, and the outputs of the first hidden layer become functionally restricted to bipolar binary values. As these activations saturate, a limited number of representations for "features" of the input vectors are available. This gradually forces "similar" inputs to activate a unique distributed representation at this layer. The larger the value of $i^{-k}$, the faster is the saturation of the activations, which, in turn, increases the sensitivity of the first-hidden-layer representations to differences among the input vectors. This increased sensitivity increases the number of unique representations at this layer, thus leading the rest of the network to reconstruct an equal number of prototypes. Hence the slope saturation parameter $\gamma$ controls cluster granularity and may be viewed as a vigilance parameter similar to that in the ART nets.

The mapping characteristics of this highly nonlinear first hidden layer also may be thought to emerge from a kind of nonlinear principal-component analysis (PCA) mechanism where unit activities are influenced by high-order statistics of the training set. The other modification to backprop is the use of exponentially damped learning coefficients. The tendency is for the network to best remember the most recently presented training data. A decaying learning coefficient helps counteract this tendency and balance the sensitivity of learning for all patterns. The learning-rate coefficient used is therefore dynamically adjusted according to

$$\rho(k) = \lambda^{k} \tag{6.12}$$

where $\lambda$ is a predefined constant less than but close to unity. As a result of this exponentially decaying learning rate, learning initially proceeds rapidly, but then the declining rate of learning produces a deemphasis of the most recently learned input vectors, which reduces "forgetting" effects and allows the repeating patterns to be learned evenly.

In the prototype-extraction phase, a dynamic net is generated by feeding the preceding trained net's output back to the input. A pass is now made over the training set, but this time no learning occurs. The primary objective of this pass is to classify the vectors in the training data and extract the prototype discovered by the network for each cluster. As each vector is presented, an output is generated and is fed back to the input of the network. This process is repeated iteratively until convergence. When the network settles into a final state, the outputs of the first hidden layer converge to a bipolar binary state. This binary state gives an intermediate distributed representation (activity pattern) of the particular cluster containing the present input. The intermediate activity pattern is mapped by the rest of the net into a real-valued activity pattern at the output layer. This output can be taken as a "nominal" representation of the underlying cluster "center," i.e., a prototype of the cluster containing the current input. Therefore, the network generates two sets of prototypes (concepts): abstract binary-valued concepts and explicit real-valued concepts. The network also supplies the user with parallel implementations of the two mappings from one concept representation to the other; the first hidden layer maps the input vectors into their corresponding abstract concepts, while the second hidden layer and the output layer implement the inverse of this mapping.

Proving the stability of this dynamic multilayer autoassociative clustering net is currently an open problem. The highly nonlinear nature of this dynamical system makes the analysis difficult. More specifically, it would be difficult to find an appropriate Liapunov function, if one exists, to prove stability. However, empirical evidence suggests a high degree of stability when the system is initialized with one of the training vectors and/or a new vector that is sufficiently similar to any one of the training vectors.

This section concludes by presenting two simulation results that illustrate the capability of the preceding autoassociative clustering net. In the first simulation, the net is used to cluster the 50 analog patterns of Figure 6-2. These patterns are repetitively presented to the net until the slopes $(\beta)$ in the first hidden layer rise to 200, according to Equation (6.11). Two experiments were performed with $\gamma = 1.0003$ and 1.0005. The network used had eight units in each of the first and second hidden layers. Figures 6-3(a) and (b) shows the learned prototypes (top row) and their associated cluster members (in associated columns) for $\gamma = 1.0003$ and $\gamma = 1.0005$, respectively $\gamma = 1.0003$ required about 350 training cycles to saturate $\beta$ at 200, while $\gamma = 1.0005$ required about 210 cycles). A typical setting for $\lambda$ in Equation (6.12) is 0.9999.

It is interesting to note that the choices of network size and parameters $i^{-k}$ and $\lambda$ were not optimized for this particular clustering problem. In fact, these parameters

were also found to be appropriate for clustering motor unit potentials in the electromyogram (EMG) signal (Wang, 1991). It is left to the reader to compare the clustering results in Figure 6-3 with those due to the ART2 net shown in Figure 6-2. Note how the level of $i^{-k}$ controls cluster granularity.

In the second simulation, the autoassociative clustering net (with comparable size and parameters as in the first simulation) is used to decompose a 10-sec. recording of an EMG signal obtained from a patient with a diagnosis of myopathy (a form of muscle disease). A segment of this signal (about 0.8 sec.) is shown in the top window in Figure 6-4. The objective here is to extract prototype motor unit potentials (MUPs) comprising this signal and to associate each noisy MUP in the signal with its correct prototype. A preprocessing algorithm detected 784 candidate MUPs (each MUP pattern is uniformly sampled and is represented by a 50-dimensional vector centered at the highest peak in the MUP signal) in the 10-sec. EMG recording shown. These detected noisy MUPs were used to train the autoassociative clustering net. The clustering results are shown in Figure 6.4.4. A total of 24 unique prototype MUPs are identified by the network. The figure shows the prototypes associated with the 11 most significant clusters in small windows; i.e., a significant number of noisy input MUPs converged to these prototypes. Examples of MUPs from each cluster are shown to the right of discovered cluster prototypes. This result is superior to those of existing automated EMG decomposition techniques. For a detailed description and analysis of this problem, the reader may consult Wang (1991).

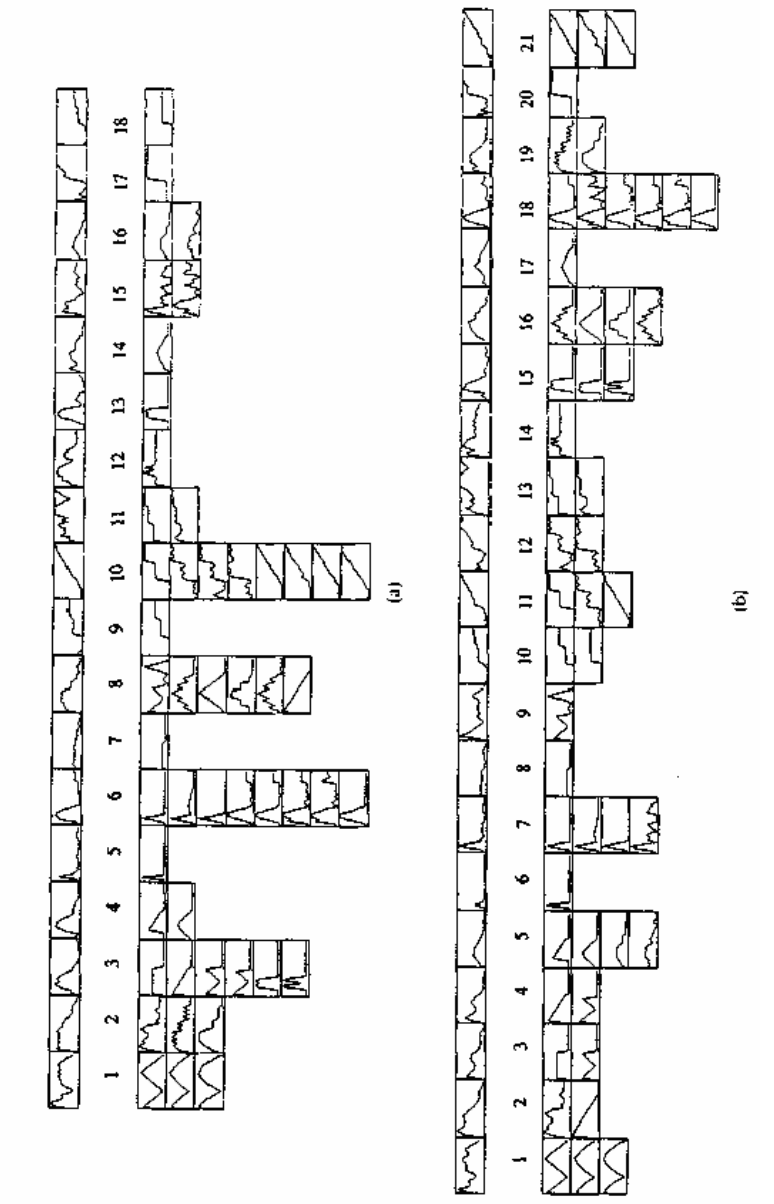


**Figure 6-3** Clusters of patterns formed by the multilayer autoassociative clustering net: (*a*) $\gamma = 1.0003$ and (*b*) $\gamma = 1.0005$ . The top row shows the learned prototype for each cluster. [From Hassoun, 1995]

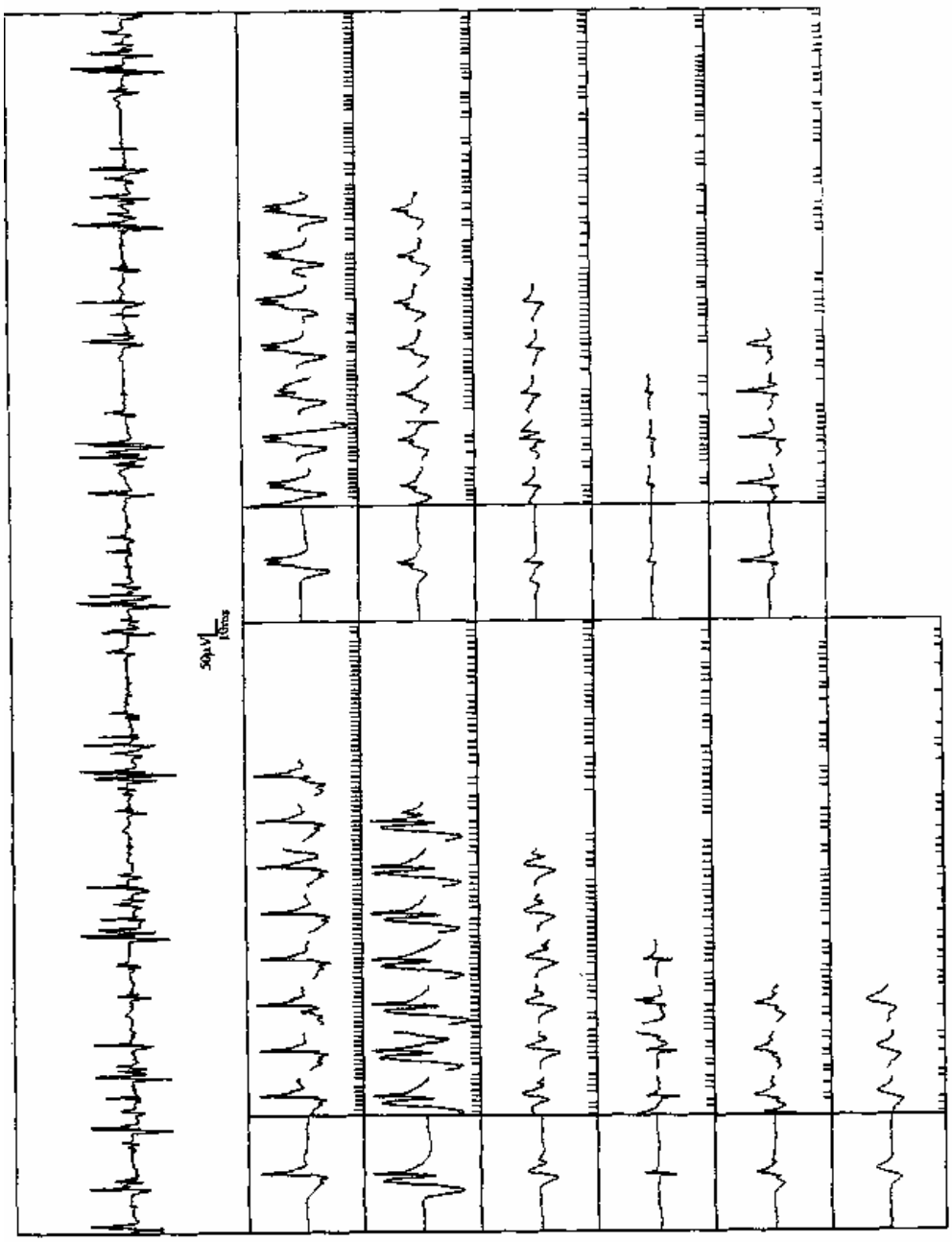

**Figure 6-4** Decomposition of an EMG signal by the autoassociative clustering net. One segment of an EMG (about 0.8s) is shown in the top window. The extracted MUP waveform prototypes are shown in small windows. Examples of classified waveforms are shown to the right.