and recognition [43, 37, 8] and on problems related to visual pattern recognition [40]. It has been found to perform well in most cases and to find good solutions to the problems posed. A demonstration of the power of this algorithm was provided by Sejnowski [43]. He trained a two-layer perceptron with 120 hidden units and more than 20,000 weights to form letter to phoneme transcription rules. The input to this net was a binary code indicating those letters in a sliding window seven letters long that was moved over a written transcription of spoken text. The desired output was a binary code indicating the phonemic transcription of the letter at the center of the window. After 50 times through a dialog containing 1024 words, the transcription error rate was only 5%. This increased to 22% for a continuation of that dialog that was not used during training.

The generally good performance found for the back propagation algorithm is somewhat surprising considering that it is a gradient search technique that may find a local minimum in the LMS cost function instead of the desired global minimum. Suggestions to improve performance and reduce the occurrence of local minima include allowing extra hidden units, lowering the gain term used to adapt weights, and making many training runs starting with different sets of random weights. When used with classification problems, the number of nodes could be set using considerations described above. The problem of local minima in this case corresponds to clustering two or more disjoint class regions into one. This can be minimized by using multiple starts with different random weights and a low gain to adapt weights. One difficulty noted with the backward-propagation algorithm is that in many cases the number of presentations of training data required for convergence has been large (more than 100 passes through all the training data). Although a number of more complex adaptation algorithms have been proposed to speed convergence [35] it seems unlikely that the complex decision regions formed by multi-layer perceptrons can be generated in few trials when class regions are disconnected.

An interesting theorem that sheds some light on the capabilities of multi-layer perceptrons was proven by Kolmogorov and is described in [26]. This theorem states that any continuous function of N variables can be computed using only linear summations and nonlinear but continuously increasing functions of only one variable. It effectively states that a three layer perceptron with $N(2N + 1)$ nodes using continuously increasing nonlinearities can compute any continuous function of N variables. A three-layer perceptron could thus be used to create any continuous likelihood function required in a classifier. Unfortunately, the theorem does not indicate how weights or nonlinearities in the net should be selected or how sensitive the output function is to variations in the weights and internal functions.

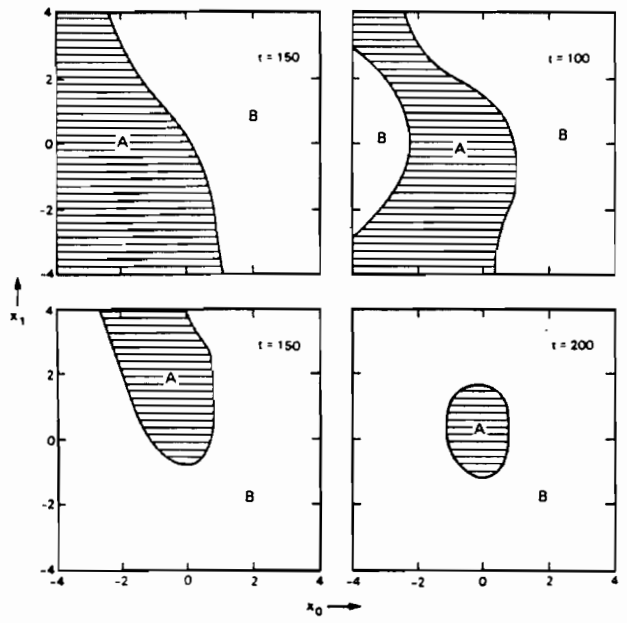

Figure 16. Decision regions after 50, 100, 150 and 200 trials generated by a two layer perceptron using the back-propagation training algorithm. Inputs from classes A and B were presented on alternate trials. Samples from class A were distributed uniformly over a circle of radius 1 centered at the origin. Samples from class B were distributed uniformly outside the circle. The shaded area denotes the decision region for class A.

KOHONEN'S SELF ORGANIZING FEATURE MAPS

One important organizing principle of sensory pathways in the brain is that the placement of neurons is orderly and often reflects some physical characteristic of the external stimulus being sensed [21]. For example, at each level of the auditory pathway, nerve cells and fibers are arranged anatomically in relation to the frequency which elicits the greatest response in each neuron. This tono-

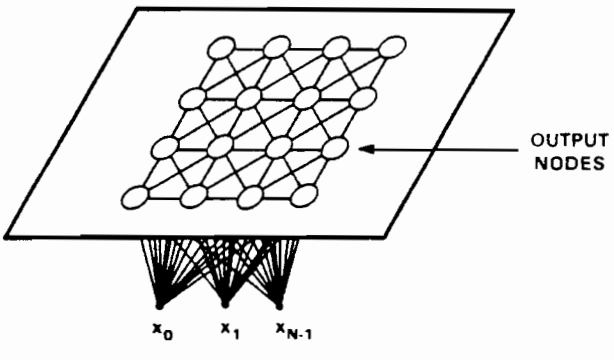

Figure 17. Two-dimensional array of output nodes used to form feature maps. Every input is connected to every output node via a variable connection weight.

topic organization in the auditory pathway extends up to the auditory cortex [33, 21]. Although much of the low-level organization is genetically pre-determined, it is likely that some of the organization at higher levels is created during learning by algorithms which promote self-organization. Kohonen [22] presents one such algorithm which produces what he calls self-organizing feature maps similar to those that occur in the brain.

Kohonen's algorithm creates a vector quantizer by adjusting weights from common input nodes to M output nodes arranged in a two dimensional grid as shown in Fig. 17. Output nodes are extensively interconnected with many local connections. Continuous-valued input vectors are presented sequentially in time without specifying the desired output. After enough input vectors have been presented, weights will specify cluster or vector centers that sample the input space such that the point density function of the vector centers tends to approximate the probability density function of the input vectors [22]. In addition, the weights will be organized such that topologically close nodes are sensitive to inputs that are physically similar. Output nodes will thus be ordered in a natural manner. This may be important in complex systems with many layers of processing because it can reduce lengths of inter-layer connections.

The algorithm that forms feature maps requires a neighborhood to be defined around each node as shown in Fig. 18. This neighborhood slowly decreases in size with time as shown. Kohonen's algorithm is described in Box 7. Weights between input and output nodes are initially set to small random values and an input is presented. The distance between the input and all nodes is computed as shown. If the weight vectors are normalized to have constant length (the sum of the squared weights from all inputs to each output are identical) then the node with the minimum Euclidean distance can be found by using the net of Fig. 17 to form the dot product of the input and the weights. The selection required in step 4 then turns into a problem of finding the node with a maximum value. This node can be selected using extensive lateral inhibition as in the MAXNET in the top of Fig. 6. Once this node is selected, weights to it and to other nodes in its neighborhood are modified to make these nodes more responsive to the current input. This process is repeated for further inputs. Weights eventually converge and are fixed after the gain term in step 5 is reduced to zero.

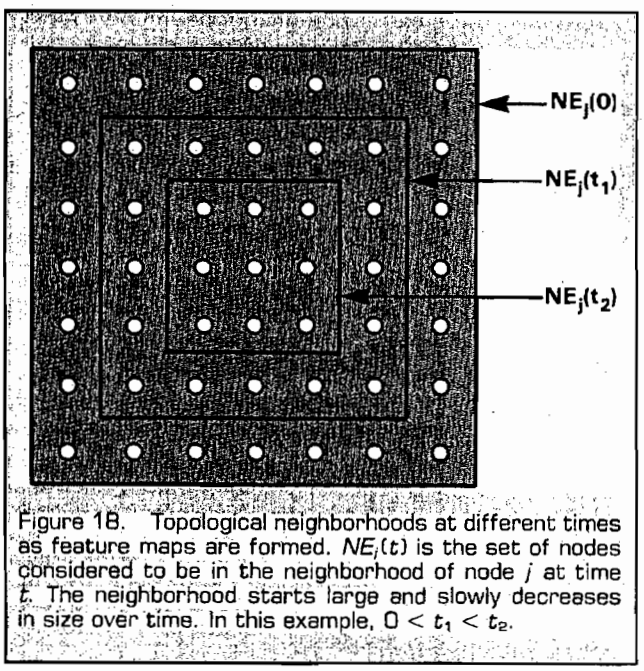

Figure 18. Topological neighborhoods at different times as feature maps are formed. $NE_j(t)$ is the set of nodes considered to be in the neighborhood of node j at time t. The neighborhood starts large and slowly decreases in size over time. In this example, $0 < t_1 < t_2$.

Box 7. An Algorithm to Produce Self-Organizing Feature Maps

Step 1. Initialize Weights

Initialize weights from N inputs to the M output nodes shown in Fig. 17 to small random values. Set the initial radius of the neighborhood shown in Fig. 18.

Step 2. Present New Input

Step 3. Compute Distance to All Nodes

Compute distances d_j between the input and each output node j using

$$d_j = \sum_{i=0}^{N-1} (x_i(t) - w_{ij}(t))^2$$

where $x_i(t)$ is the input to node i at time t and $w_{ij}(t)$ is the weight from input node i to output node j at time t.

Step 4. Select Output Node with Minimum Distance

Select node j^* as that output node with minimum d_j.

Step 5. Update Weights to Node j^* and Neighbors

Weights are updated for node j^* and all nodes in the neighborhood defined by $NE_{j^*}(t)$ as shown in Fig. 18. New weights are

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i(t) - w_{ij}(t))$$

For $j \in NE_{j^*}(t)$ $\quad 0 \le i \le N-1$

The term $\eta(t)$ is a gain term $(0 < \eta(t) < 1)$ that decreases in time.

Step 6. Repeat by Going to Step 2

An example of the behavior of this algorithm is presented in Fig. 19. The weights for 100 output nodes are plotted in these six subplots when there are two random independent inputs uniformly distributed over the region enclosed by the boxed areas. Line intersections in these plots specify weights for one output node. Weights from

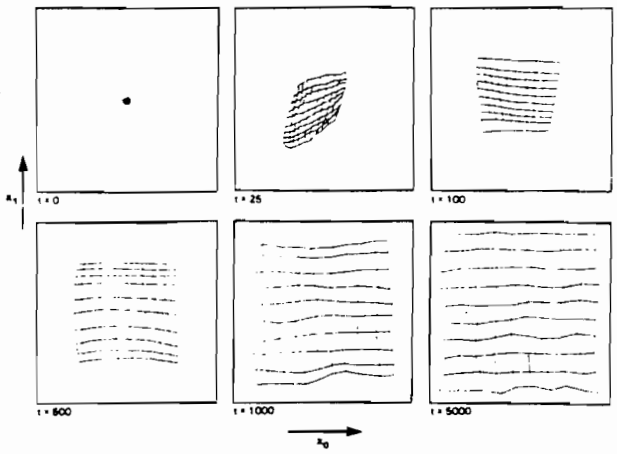

Figure 19. Weights to 100 output nodes from two input nodes as a feature map is being formed. The horizontal axis represents the value of the weight from input x_0 and the vertical axis represents the value of the weight from input x_1. Line intersections specify the two weights for each node. Lines connect weights for nodes that are nearest neighbors. An orderly grid indicates that topologically close nodes code inputs that are physically similar. Inputs were random, independent, and uniformly distributed over the area shown.

input x_0 are specified by the position along the horizontal axis and weights from input x_1 are specified by the position along the vertical axis. Lines connect weight values for nodes that are topological nearest neighbors. Weights start at time zero clustered at the center of the plot. Weights then gradually expand in an orderly way until their point density approximates the uniform distribution of the input samples. In this example, the gain term in step 5 of Box 7 was a Gaussian function of the distance to the node selected in step 4 with a width that decreased in time.

Kohonen [22] presents many other examples and proofs related to this algorithm. He also demonstrates how the algorithm can be used in a speech recognizer as a vector quantizer [23]. Unlike the Carpenter/Grossberg classifier, this algorithm can perform relatively well in noise because the number of classes is fixed, weights adapt slowly, and adaptation stops after training. This algorithm is thus a viable sequential vector quantizer when the number of clusters desired can be specified before use and the amount of training data is large relative to the number of clusters desired. It is similar to the *K*-means clustering algorithm in this respect. Results, however, may depend on the presentation order of input data for small amounts of training data.

INTRODUCTORY REFERENCES TO NEURAL NET LITERATURE

More detailed information concerning the six algorithms described above and other neural net algorithms can be found in [3, 7, 15, 18, 19, 20, 22, 25, 32, 39, 40]. Descriptions of many other algorithms including the Boltzmann machine and background historical information can be found in a recent book on parallel distributed processing edited by Rumelhart and McClelland [41]. Feldman [9] presents a good introduction to the connectionist philosophy that complements this book. Papers describing recent research efforts are available in the proceedings of the 1987 Conference on Neural Networks for Computing held in Snowbird, Utah [6]. Descriptions of how the Hopfield net can be used to solve a number of different optimization problems including the traveling salesman problem are presented in [20, 45]. A discussion of how content-addressable memories can be implemented using optical techniques is available in [1] and an introduction to the field of neurobiology is available in [21] and other basic texts.

In addition to the above papers and books, there are a number of neural net conferences being held in 1987. These include the "1987 Snowbird Meeting on Neural Networks for Computing" in Snowbird, Utah, April 1–5, the "IEEE First Annual International Conference on Neural Networks" in San Diego, California, June 21–24, and the "IEEE Conference on Neural Information Processing Systems—Natural and Synthetic" in Boulder, Colorado, November 8–12. This last conference is cosponsored by the IEEE Acoustics, Speech, and Signal Processing Society.

CONCLUDING REMARKS

The above review provides an introduction to an interesting field that is immature and rapidly changing. The six nets described are common components in many more complex systems that are under development. Although there have been no practical applications of neural nets yet, preliminary results such as those of Sejnowski [43] have demonstrated the potential of the newer learning algorithms. The greatest potential of neural nets remains in the high-speed processing that could be provided through massively parallel VLSI implementations. Several groups are currently exploring different VLSI implementation strategies [31, 13, 42]. Demonstrations that existing algorithms for speech and image recognition can be performed using neural nets support the potential applicability of any neural-net VLSI hardware that is developed.

The current research effort in neural nets has attracted researchers trained in engineering, physics, mathematics, neuroscience, biology, computer sciences and psychology. Current research is aimed at analyzing learning and self-organization algorithms used in multi-layer nets, at developing design principles and techniques to solve dynamic range and sensitivity problems which become important for large analog systems, at building complete systems for image and speech and recognition and obtaining experience with these systems, and at determining which current algorithms can be implemented using neuron-like components. Advances in these areas and in VLSI implementation techniques could lead to practical real-time neural-net systems.