and recognition [43, 37, 8] and on problems related to
to visual pattern recognition [40]. It has been found to per-
form 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 allow-
ing 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 mini-
mized 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 pro-
posed to speed convergence [35] it seems unlikely that
the complex decision regions formed by multi-layer per-
ceptrons 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 com-
puted 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 non-
linearities 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 indi-
cate 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.

Kohonen's Self-Organizing Feature Maps

One important organizing principle of sensory path-
ways 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 tomo-

![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.]

![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 clusters 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.

**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

$$d_j = \sum_{k=1}^{N-1} (x_k(t) - w_{k,j}(t))^2$$

where $x_k(t)$ is the input to node $k$ at time $t$ and $w_{k,j}(t)$ is the weight from input node $k$ 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 $N(1)$ as shown in Fig. 18. New weights are

$$w_{k,j}(t + 1) = w_{k,j}(t) + \eta(t)(x(t) - w_{k,j}(t))$$

For $j \in N(1)$

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
TABLE I
$X_A$ ANIMAL (RITTER AND KOHONEN [6])

<table>
<thead>
<tr>
<th>Animal</th>
<th>$g$</th>
<th>$e$</th>
<th>$w$</th>
<th>$i$</th>
<th>$l$</th>
<th>$o$</th>
<th>$e$</th>
<th>$a$</th>
<th>$o$</th>
<th>$g$</th>
<th>$f$</th>
<th>$d$</th>
<th>$c$</th>
<th>$i$</th>
<th>$r$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>medium</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>big</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Has</th>
<th>2 legs</th>
<th>4 legs</th>
<th>hair</th>
<th>hooves</th>
<th>mane</th>
<th>feathers</th>
<th>hunt</th>
<th>run</th>
<th>to fly</th>
<th>swim</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 legs</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4 legs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hair</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hooves</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mane</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>feathers</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hunt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>run</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>to fly</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>swim</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Fig. 1. The SOFM architecture with iterate number (t) suppressed.