Additional Applications of Kohonen Networks

Color Quantization Problem (discussed earlier)

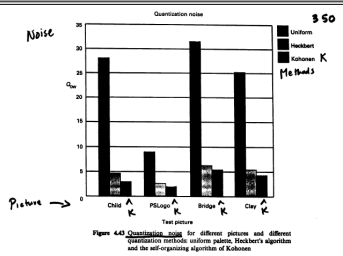


Figure 4.43 Quantization noise for different pictures and different quantization methods: uniform palette, Heckbert's algorithm and the self-organizing algorithm of Kohonen

Heckbert's median cut algorithm "Color Quantization by Dynamic Programming and Principal Analysis" by Xiaolin Wu in ACM Transactions on Graphics, Vol. 11, No. 4, October 1992, 348-372.

Animal Similarity Expt.

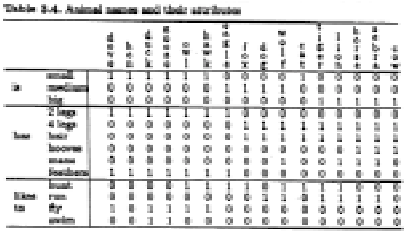


Table 3.4. Animal names and their attributes

Animal Similarity Expt.

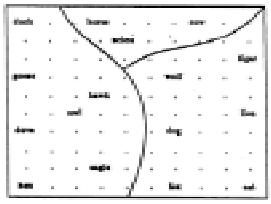


Fig. 3.22. After the network had been trained with inputs describing attributes sets from Table 3.4, the map was calibrated by the columns of Table 3.4 and labeled correspondingly. A grouping according to similarity has emerged

Semantic Map Expt.

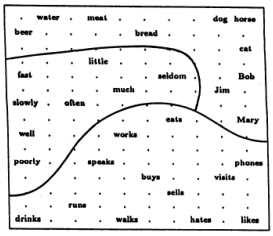


Fig. 7.11. "Semantic map" obtained on a network of 10 × 15 cells after 2000 presentations of word-context-pairs derived from 10,000 random sentences of the kind shown in Fig. 7.8. Nouns, verbs and adverbs are segregated into different domains. Within each domain a further grouping according to aspects of meaning is discernible

Tree Data Structure Expt.

- Encode a 5-dimensional tree as shown:

```
ABCDE
  F
  G
  H K L M N O P Q R
  I     S     W
  J     T     X 1 2 3 4 5 6
        U     Y
        V     Z
```

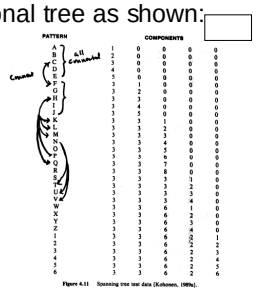


Figure 4.11 Spanning tree test data [Kohonen, 1989a].

Tree Data Structure Expt.

- Use a 2D imposed structure, with 10 x 7 neurons
- Results:
  Branches of tree tend to align in rows or columns of the array.

  5D → 2D mapping

Tree Data Structure Expt.

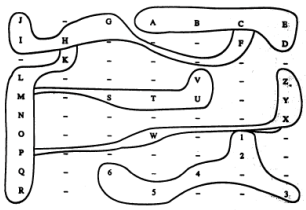


Figure 4.16 Results after another 75 iterations with $R = 0$.

Compare Original 5DTree

```
ABCDE
  F
  G
H K L M N O P Q R
I     S     W
J     T     X 1 2 3 4 5 6
      U     Y
      V     Z
```

Tree Data Structure Expt. Using Hexagonal Array

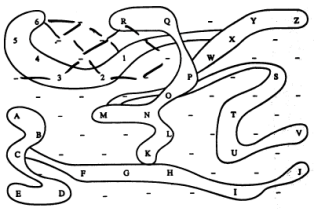


Figure 4.17 Results of spanning tree example using hexagonal array.

Compare Original 5DTree

```
ABCDE
  F
  G
H K L M N O P Q R
I     S     W
J     T     X 1 2 3 4 5 6
      U     Y
      V     Z
```

Function Approximation Expt.

- Assume 1-dimensional inputs for simplicity (not required in general).
- Represent pairs y = f(x) as (x, y).
- Learn pairs by a 2-D Kohonen net.

A Possible Aberration

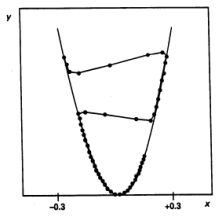


Figure 4.63 Representation of 1000 pairs of argument and function values of the function $y = 10x^2$ by two-dimensional weight vectors (dots) in a one-dimensional neural net of fifty neurons. Input argument values in $[-0.3, +0.3]$

Master-Slave Technique

- A technique known as “master-slave” can reduce the likelihood of an aberration as shown previously.
  - Select **winners** based only on x, not on the pair (x, y). The weight components corresponding to x are called the **master weights**, and others are called **slave weights.**
  - Adapt both master and slave.

No Aberration

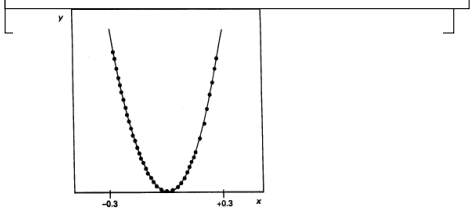


Figure 4.66 The master–slave method. The representation of 1000 pairs of argument and function values of the function $y = 10x^2$ by two-dimensional master–slave weight vectors (dots) in a one-dimensional neural net of fifty neurons. Input argument values in $[-0.3, +0.3]$

Example: Robot Hand-Eye Coordination

- Want robot to coordinate its hand with what it sees on the monitor.
- Robot arm should be able to place an object given position identified in **view**.
- The problem is how to set the arm's angles (called "inverse kinematics").

Robot Hand-Eye

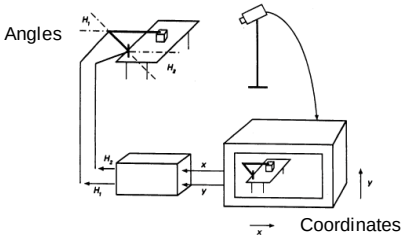


Figure 4.68 Simple outline of robot arm control

Master-Slave

- The view is the master.
- The hand is the slave.
- Winner is determined w.r.t. the view; both view and hand are adapted.
- Example:
  - 20x20 neuron map, 2D overlay
  - 4 weights per neuron (2 for master/view and 2 for slave/angles)

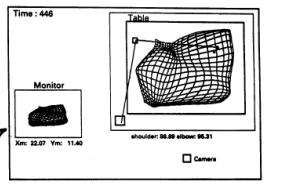


Figure 4.70 Representation of table coordinates by the two-dimensional master–slave weight vectors (dots in the right-hand figure) in a two-dimensional net of 20×20 neurons after 446 training examples. Weight vectors are connected with a line if they belong to neighbouring neurons

Positions on the monitor,
not what the monitor is seeing.

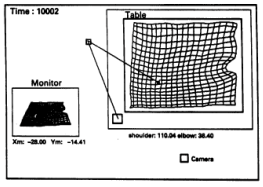
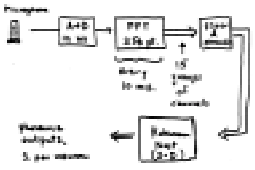


Figure 4.71 Representation of table coordinates by the two-dimensional master–slave weight vectors (dots in the right-hand figure) in a two-dimensional net of 20×20 neurons after 10 002 training examples. Weight vectors are connected with a line if they belong to neighbouring neurons

Phonetic Typewriter

- Objective: Spoken words →phonemes
- Languages: Finnish, Japanese
- Claimed ≥ 92% accuracy in 10 mins. of training.



Resulting Phonotopic Map

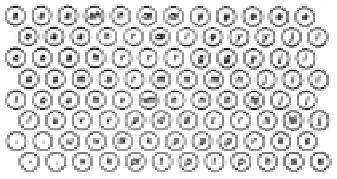


Figure 7.9 This figure is a phonotopic map with the neurons, shown as circles, and the phonemes to which they learned to respond. A double label means that the node responds to more than one phoneme. Some phonemes—such as the plosives represented by *k*, *p*, and *t*—are difficult for the network to distinguish and are most prone to misclassification by the network. *Source:* Reprinted with permission from Teuvo Kohonen, "The neural phonetic typewriter," IEEE Computer, March 1988. © 1988 IEEE.

Problems Unsuitable for Kohonen Nets

- Kohonen nets work by clustering
- Nearby data points are expected to behave similarly, e.g. have similar outputs.
- Parity-like problems such as the XOR do not have this property. The would be unstable for solution by a Kohonen net.

Resulting Phonotopic Map

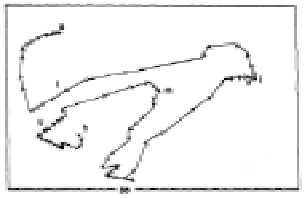


Figure 7.10 This illustration shows the sequence of responses from a phonotopic map resulting from the spoken Finnish word humppila. (Do not bother to look up the meaning of this word in your Finnish-English dictionary: humppila is the name of a place.) *Source:* Reprinted with permission from Teuvo Kohonen, "The neural phonetic typewriter," IEEE Computer, March 1988. © 1988 IEEE.

Competitive Learning with Explicit Output Classification

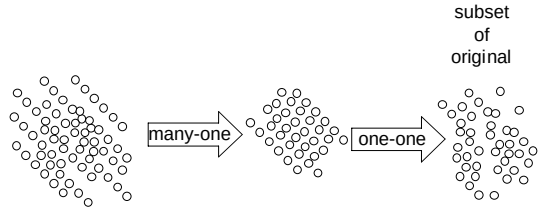
Learning Vector-Quantization

Counterpropagation

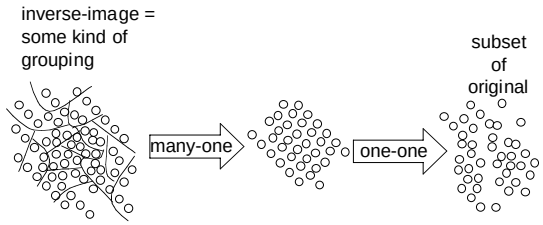
Vector Quantization

- VQ is a technique used to reduce the dimensionality of data.
- The original data is a set of vectors, of dimension n, say.
- The vectors are mapped into a smaller set of **codebook values**.
- The codebook values are used for storage or transmission (some information content is lost).
- The codebook values are re-translated into values close to the original data values (will not be exact).

Codebook

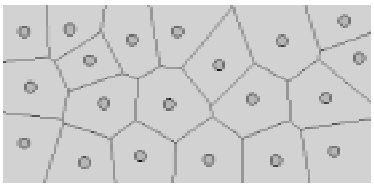


Codebook



Voronoi Regions

Typically codebook code representatives are centers of ***Voronoi Regions*** in the data space. Voronoi regions are sets of points **closest** to one representative vs. another. Below each polygon is a Voronoi region, with representatives as dots.



closest ⇒ error is minimized

Chicken vs. Egg

- Which comes first, the Voronoi regions or the representatives?
- Initially we just have a set of points with neither.
- We specify the maximum size of the codebook (assume the number of points is larger).

Chicken vs. Egg

- Choose an initial set of representatives by some method.
- Iterate the following:
  - For each point, find its closest representative; this determines a set of Voronoi regions.
  - Compute the *centroids* of the regions and use them as the new representatives.
- Until the desired error bound is reached.
- Note: The representatives need not be actual data points.

Termination Condition

- Compute MSE =
$$\frac{\Sigma\{\text{distance-from-rep}(n)^2 \mid n \text{ in region}\}}{\#\text{regions}}$$

  where the sum is over all regions.

- We want MSE to be below a certain bound.

Computational Cost

- $O(n^2)$ per iteration, where n is the number of points.

Application

- Clustering methods such as the preceding can be used to set the centers in a Radial Basis Function network.

Learning Vector-Quantization

Learning Vector Quantization (LVQ)

- Combine competitive learning with supervision
- Competitive learning achieves clusters
- Assign a **class** (or output value) to each cluster
- Reinforce cluster representative (a neuron) when it **classifies** input in the desired class:
  - **Positive reinforcement**: pull the neuron weights toward the input.
  - **Negative reinforcement**: push the weights away.

Learning Vector Quantization

Training examples:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

If the input pattern is classified ***correctly***, then move the *winning* weight toward the input vector according to the Kohonen rule.

$$_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)) \qquad a^2_{k^*} = t_{k^*} = 1$$

If the input pattern is classified ***incorrectly***, then move the *winning* weight *away* from the input vector.

$$_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)) \qquad a^2_{k^*} = 1 \neq t_{k^*} = 0$$

Learning Vector Quantization

- The classification mapping from selection of cluster representative to output can be accomplished by a second “layer”.
- This second layer is essentially a matrix multiply, so can be represented by neural weights as usual. These weights may be pre-assigned and fixed.

LVQ xor Example

Example patterns with targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \left\{\mathbf{p}_2 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \left\{\mathbf{p}_3 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\} \left\{\mathbf{p}_4 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$$

Competitive layer weights:

$$\mathbf{W}^1(0) = \begin{bmatrix}({}_1\mathbf{w}^1)^T\\({}_2\mathbf{w}^1)^T\\({}_3\mathbf{w}^1)^T\\({}_4\mathbf{w}^1)^T\end{bmatrix} = \begin{bmatrix}0.25 & 0.75\\0.75 & 0.75\\1 & 0.25\\0.5 & 0.25\end{bmatrix}$$

Classification layer weight (fixed):

$$\mathbf{W}^2 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}$$

Iteration 1, first layer

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left(\begin{bmatrix}-\|{}_1\mathbf{w}^1 - \mathbf{p}_1\|\\-\|{}_2\mathbf{w}^1 - \mathbf{p}_1\|\\-\|{}_3\mathbf{w}^1 - \mathbf{p}_1\|\\-\|{}_4\mathbf{w}^1 - \mathbf{p}_1\|\end{bmatrix}\right)$$

$$\mathbf{a}^1 = \mathbf{compet}\left(\begin{bmatrix}-\|\begin{bmatrix}0.25 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\|\\-\|\begin{bmatrix}0.75 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\|\\-\|\begin{bmatrix}1.00 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\|\\-\|\begin{bmatrix}0.50 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\|\end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix}-0.354\\-0.791\\-1.25\\-0.901\end{bmatrix}\right) = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

Iteration 1, second layer

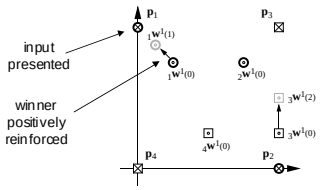
$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}$$

This is the correct class, therefore the weight vector is moved toward the input vector.

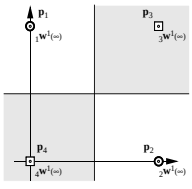
$${}_1\mathbf{w}^1(1) = {}_1\mathbf{w}^1(0) + \alpha(\mathbf{p}_1 - {}_1\mathbf{w}^1(0))$$

$${}_1\mathbf{w}^1(1) = \begin{bmatrix}0.25\\0.75\end{bmatrix} + 0.5\left(\begin{bmatrix}0\\1\end{bmatrix} - \begin{bmatrix}0.25\\0.75\end{bmatrix}\right) = \begin{bmatrix}0.125\\0.875\end{bmatrix}$$

Iteration 1 illustrated



Final Outcome after many iterations



Variant: LVQ2

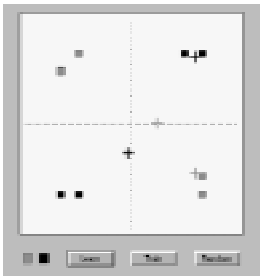
- If the winning neuron in the hidden layer **correctly** classifies the current input, the response is the same as in LVQ1.
- If the winning neuron in the hidden layer **incorrectly** classifies the current input, we move its weight vector away from the input vector, as in LVQ, but
- ***also*** move the weights of the closest neuron to the input vector that **does** classify the input properly toward the input vector.

Matlab LVQ demos

- nnd14lv1, nnd14lv2

Blue vs. black represent class of neuron, desired class of input sample.

Selected input sample changes to green if correctly classified, red if not; weights adjusted.



Counterpropagation Networks

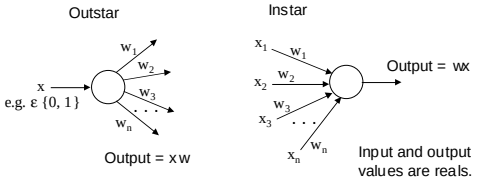
Counterpropagation Networks

- Invented by Robert Hecht-Nielson, founder of HNC Inc.
- Consists of **two** opposing networks, one for learning a **function**, the other for learning its **inverse**.
- Each network has two layers:
  - A Kohonen first layer that clusters inputs.
  - An "outstar" second layer to provide the output values for each cluster.

Outstar and Instar (due to Stephen Grossberg)

- An **instar** responds to a single input.
- An **outstar** produces a single (multi-dimensional) output d when stimulated with a binary value x.

Outstar and Instar



Biologically, outstar would be synaptic weights, while instar would have dendritic ones.
However, it is common to refer to weights as "synaptic".

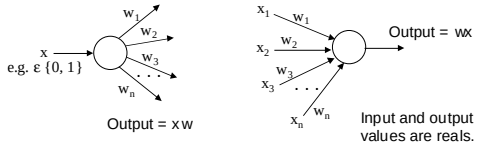
Outstar and Instar

- Outstar learning rule:

  $\Delta w = \alpha\, x\, (d - w)$
  (d = desired)

- Instar learning rule:

  $\Delta w = \alpha (x - w)\, d$
  (d = desired)



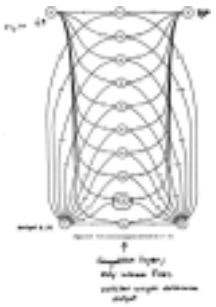
Outstar and Instar

- Variations are possible, e.g. adding weight-decay.

Counterpropagation Operation

- An outstar neuron is associated with each cluster representative.
- Given an input, the winner is found.
- The outstar is then stimulated to give the output.

Counterpropagation Example Reciprocal Network



Counterpropagation Notes

- Since these networks operate by **recognizing input patterns** in the first layer, one would generally use ***lots*** of neurons in this layer.

Counterpropagation vs. LVQ

- Both combine Supervise and Unsupervised learning.
- LVQ is used for classification; Counterprop can be used for function computation
- LVQ has one training phase; Counterprop has two.
- LVQ output weights are fixed throughout; Counterprop weights are trained in a second phase.