Self-Organization

In this chapter we consider unsupervised learning by self-organization.

For these models, a correct output cannot be defined a priori.

Therefore, a numerical measure of the magnitude of the mapping error cannot be used to derive a learning (weight adaptation) technique.

The Brain as a Self-Organizing, Adaptive System

The brain adapts its structure in a self-organized fashion by changing the interconnections among neurons:

- adding neurons and/or connections
- removing neurons and/or connections
- strengthening:
  - increasing the number of transmitters released at synapses
  - increase the size of the synaptic cleft
  - forming new synapses.
Donald Hebb (1949) explicitly stated conditions that allow changes at the synaptic level to reflect learning and memory:

"When an axon of cell A is near enough to excite a cell B, and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one of both cells such that A's efficiency, as one of the cells firing [with] cell B is increased."

**Charting Input Space**

When a self-organizing network is used, an input vector is presented at each step. These input vectors constitute the "environment" of the network. Each new input results in an adaptation of the parameters of the network. If such modifications are correctly controlled, the network can build an *internal representation of the environment*.

> Mapping from Input to Output Space $f: A \rightarrow B$

![Figure 1. Mapping from input to output space](image)

If an input space is to be processed by a neural network, the first issue of importance is the *structure of this space*. 
A neural network with real inputs computes a function \( f: A \rightarrow B \), from an input space \( A \) to an output space \( B \).

The region where \( f \) is defined can be covered by a network (SOF) in such a way that only one unit in the network fires when an input vector from a particular region is selected (for example \( a_1 \)).

**Topology Preserving Maps in the Brain**

Many structures in the brain have a linear or planar topology, that is, they extend in one or two dimensions.

Sensory experience, however, is multidimensional.

Example: Perception

- colour: three different light receptors
- position of objects
- texture of objects
- …

How do the planar structures in the brain manage to process such multidimensional signals?

How is the multidimensional input projected to the two-dimensional neuronal structures?

**Mapping of the Visual Field on the Cortex**

The visual cortex is a well-studied region in the posterior part of the human brain. The visual information is mapped as a two-dimensional projection on the cortex.
Two important phenomena can be observed in the above diagram:

- Neighbouring regions of the visual field are processed by neighbouring regions in the cortex.
- The surface of the visual cortex reserved for processing signals from the center of the visual field are processed in more detail and with higher resolution. Visual acuity increases from the periphery to the center.

⇒ topologically ordered representation of the visual field

- The Somatosensory and Motor Cortex

The human cortex also establishes a topologically ordered representation of sensations coming from other organs.
The figure shows a slice of two regions of the brain:

- the somatosensory cortex, responsible for processing mechanical inputs,
- the motor cortex, which controls the voluntary movement of different body parts.

Both regions are present in each brain hemisphere and are located contiguous to each other.

The region in charge of signals from the arms, for example, is located near to the region responsible for the hand. The spatial relations between the body parts are preserved as much as possible.

The same phenomenon can be observed in the motor cortex.
Self-Organizing Feature Maps (SOFs)

Kohonen Networks

The best-known and most popular model of self-organizing networks is the topology-preserving map proposed by Teuvo Kohonen (following ideas developed by Rosenblatt, von der Malsburg, and other researchers).

Kohonen's networks are arrangements of computing nodes in one-, two-, or multi-dimensional lattices.

The units have lateral connections to several neighbours.
General Structure of Kohonen Networks

Kohonen Units

A Kohonen unit computes the Euclidean distance between an input vector $\vec{x}$ and its weight vector $\vec{w}$:

$$\text{output} = ||\vec{x} - \vec{w}||$$

This new definition of neuron excitation is more appropriate for topological maps. Therefore, we diverge from sigmoidal activation functions.
One-dimensional Lattice

Consider the problem of charting an $n$-dimensional space using a one-dimensional chain of Kohonen units.

The units are all arranged in sequence and are numbered from 1 to $m$.

Each unit $i$ receives the $n$-dimensional input $\mathbf{x}$ and computes the corresponding excitation $\|\mathbf{x} - \mathbf{w}_i\|$.

The objective is that each unit learns to specialize on different regions of the input space.

**Lattice Configurations and Neighbourhood Functions**

Kohonen learning uses a neighbourhood function $\Phi$, whose value $\Phi(i, k)$ represents the strength of the coupling between unit $i$ and unit $k$ during the training process.

A simple choice is defining

$$
\Phi(i, k) = \begin{cases} 
1 & |i - k| \leq r \\
0 & |i - k| > r
\end{cases}
$$
Two-dimensional Lattice

Cylinder Neighbourhood

\[ h_{\text{cylinder}}(z, d) = \begin{cases} 1 & ; z < d \\ 0 & ; \text{otherwise} \end{cases} \]

```
h_{\text{cylinder}}[z_, d_] := 1 /; z < d
h_{\text{cylinder}}[z_, d_] = 0;
```
Plot3D[\[h_{cylinder}\] \[\sqrt{x^2 + y^2}\], 1.0],
\{(x, -2, 2), (y, -2, 2), PlotPoints \rightarrow 100, Mesh \rightarrow False\];

\[\text{Cone Neighbourhood}\]

\[h_{cone}[z_, d_] := 1 - \frac{z}{d}; \quad z < d\]
\[h_{cone}[z_, d_] = 0;\]
Plot3D[hcone [\( \sqrt{x^2 + y^2} \), 1.0], {x, -2, 2},
{y, -2, 2}, PlotPoints \to 50, Mesh \to False];

- Gauss Neighbourhood

\[ h_{\text{gauss}} [z, d] := e^{-\left(\frac{z}{d}\right)^2} \]
Plot3D\left[h_{\text{gauss}}\left[\sqrt{x^2+y^2}, 1.0\right], \{x, -2, 2\}, \{y, -2, 2\}, \text{PlotPoints} \to 50, \text{Mesh} \to \text{False}\right];

Table\left[\text{Plot3D}\left[h_{\text{gauss}}\left[\sqrt{x^2+y^2}, d\right], \{x, -2, 2\}, \{y, -2, 2\}, \text{PlotPoints} \to 50, \text{Mesh} \to \text{False}\right], \{d, 0.1, 2, 0.1\}\right];
Cosine Neighbourhood

\[ h_{\text{cosine}}[z, d] := \cos\left[ \frac{z}{d} \right] \quad / \quad z < d \]
\[ h_{\text{cosine}}[z, d] = 0; \]

\[ \text{Plot3D}[h_{\text{cosine}}[\sqrt{x^2 + y^2}, 1.0], \]
\[ \{x, -2, 2\}, \{y, -2, 2\}, \text{PlotPoints} \rightarrow 50, \text{Mesh} \rightarrow \text{False}] ; \]
SOF Learning Algorithm

The Kohonen Learning Algorithm

Start:

The $n$-dimensional weight vectors $\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_m$ of the $m$ computing units are selected at random.

An initial radius $r$, a learning constant $\eta$, and a neighbourhood function $\Phi$ are selected.

Step 1:

Select an input vector $\xi$ using the desired probability distribution over the input space.
Step 2:
The unit $k$ with the maximum excitation is selected, i.e., the unit for which the distance between $\bar{\xi}_i$ and $\xi$ is minimal:
\[
\|\xi - \bar{\xi}_k\| \leq \|\xi - \bar{\xi}_i\| \text{ for all } i = 1, \ldots, m.
\]

Step 3:
The weight vectors are updated using the neighbourhood function and the update rule
\[
\bar{w}_i := \bar{w}_i + \eta \cdot \Phi(i, k) \cdot (\xi - \bar{w}_i) \text{ for } i = 1, \ldots, m.
\]

Step 4:
Stop if the maximum number of iterations has been reached. Otherwise, modify $\eta$ and $\Phi$ as scheduled and continue with step 1.

Illustrating Euclidean Distance

A simple way to compute distances between vectors in 2D space is through the dot product of the normalized vectors $\bar{v}^* = \frac{\bar{v}}{|\bar{v}|}$, $\bar{w}^* = \frac{\bar{w}}{|\bar{w}|}$:

\[
\bar{v}^* \cdot \bar{w}^* = |\bar{v}| \cdot |\bar{w}| \cdot \cos(\bar{v}, \bar{w}) = \cos(\bar{v}, \bar{w})
\]
Figure 6. Distance of vectors through the dot product

- Adjusting Weight Vectors in 2D Space

Figure 7. Illustration of a learning step in Kohonen networks
Clustering of Vectors

Figure 8. Clustering of vectors for a particular input distribution

Elasticity

During the training phases the neighbourhood function can change its radius or its "elasticity", such that the further learning progresses the less changes are made to the network (compare simulated annealing).
Figure 9. Function for adjusting elasticity over time.
Applications

Simple Maps

- Mapping a Chain to a Triangle

Figure 10. Mapping a chain of neurons to a triangle.
Mapping a Chain to a Square

"Peano Curve": Mapping a chain of neurons to a square. (a) Randomly selected initial state; (b) after 200 iterations; (c) after 50000 iterations; (d) after 100000 iterations.
Mapping a Square with a Two-Dimensional Lattice

Mapping a square onto a two-dimensional lattice. The diagram on the right shows some overlapped iterations of the learning process. The last diagram is the final state after 10,000 iterations.

Figure 12
Planar Network with a Knot

A planar network with a know. The knot formed during the training process. If the placticity of the network has reached a low level, the knot will not be undone by any further training.

2D Map of a 3D Region

Two-dimensional network used to chart a three-dimensional box. The network extends in $x$ and $y$ dimension and folds in the $z$ direction.
Figure 15. Diagram of eye dominance in the visual cortex. Black stripes represent one eye, white stripes represent the other eye.
Traveling Salesman

Figure 16: Solving the TSP problem for 30 cities with a chain of Kohonen units. The iterations shown are 5000, 7000, and 10000.
Visumotor Coordination of Robot Arms

Figure 17. Schematic representation of the motor-coordination learning problem. The two 2D-coordinates of the target object (as seen from camera 1 and 2) are fed into the 3D Kohonen lattice. Neuron $s$, as the representative of the selected region, calculates parameters to navigate the robot arms to the location of the target object.
Figure 18. The 3D Kohonen network at the beginning (top), after 2000 iterations (middle), and after 6000 learning steps (bottom).
References


