**Self-Organising Systems**

Learning in a machine learning system may be **supervised** or **unsupervised**.

- **supervised**: system is typically trained on cases for which "the answer" is known, i.e. on pairs consisting of an input pattern and a desired output pattern.
- **unsupervised**: system is given data and told to find interesting regularities or groupings in it. No external teacher

### Some Principles of Self-Organisation

Self-organised learning in a neural system consists of repeatedly modifying the synaptic weights of the network in response to activation patterns and in accordance with prescribed rules, until a final, stable configuration develops.

The fact that this strategy can lead to useful nets confirms Turing's (1952) observation:

*Global order can arise from local interactions*

**activity** - refers to neuron output values in response to inputs  
**connectivity** - refers to synaptic weights between neurons

Self-organisation is about the interaction between activity and connectivity. The feedback between changes in weights and changes in activity patterns must be positive in order to achieve self-organisation.

### Four Principles of Self-Organisation

1. Modifications in synaptic weights tend to self-amplify.
2. Limitation of resources leads to competition among synapses and therefore the selection of the most vigorously growing synapses at the expense of others.
3. Modifications in synaptic weights tend to cooperate.
4. Order and structure in the activation patterns represent redundant information that is acquired by the neural network in the form of knowledge, which is a necessary prerequisite to self-organised learning.
Self-Organising Maps (SOMs)

The human brain is organised in many places so that sensory inputs are represented by topologically ordered maps - thus for tactile sensations, the cortical area that responds to touching with the left index finger will be near that for the left middle finger, and so on. This makes good engineering sense, as it means connections between related pieces of information are likely to be short.

The locations in the sensory cortex and motor cortex that correspond to different parts of the body have in fact been mapped¹:

¹ Images from http://calder.med.miami.edu/pointis/tbiprov/NURSING/over3.html

Self-Organising Maps (SOMs) 2

Self-organising maps are inspired by this biological role-model. The neurons are placed at the nodes of an $n$-dimensional lattice (usually $n = 1$ or 2).

Figure 9.1(b): Kohonen Model

The notion of neighbour is important in self-organising maps. Thus the winning neuron in the diagram above has eight immediate neighbours.

Self-Organising Maps (SOMs) 3

Figure 9.1(b) showed just one input neuron, but in practice there would be several or many input neurons, and their collective activations are termed an input pattern. A typical input pattern will consist of a "spot" of activity against a quiet background.

Multi-neuron 2-D Kohonen SOM Network

Multi-neuron 1-D Kohonen SOM Network with $m$ input neurons, $l$ map neurons

The goal of the SOM is to transform an input pattern of arbitrary dimension into a 1-D or 2-D discrete map, in a topologically ordered way.

Teuvo Kohonen

Kohonen is a Finnish researcher, now an emeritus professor (born 1934), who invented the Self-Organising Map algorithm (aka Kohonen map), and a prominent figure in neural network research for many years.

He worked on neural networks throughout the 1970s, when neural network research was supposedly in eclipse, and published his work on the SOM in 1982 (cf. 1985 or so for backprop).

He also produced the Learning Vector Quantisation (LVQ) algorithm, which we will study later.

Self-organisation Notes © Bill Wilson, 2013
Components of SOM Algorithm

SOM algorithm initialisation - set all weights to small random values.

Three processes:

1. **Competitive Process.** For each input pattern, each neuron computes its value for a discriminant function. The neuron with the biggest discriminant function value is the winner.

2. **Cooperative Process.** The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, and the neurons in the neighbourhood may then cooperate.

3. **Synaptic Adaptation.** Excited neurons may increase their values of the discriminant function in relation to the current input pattern (or a similar one) by weight adjustments.

1. **Competitive Process**

Let \( m \) denote the dimension of the input pattern

\[ \mathbf{x} = [x_1, x_2, ..., x_m]^T \]

The weight vector for each of the \( l \) neurons in the SOM also has dimension \( m \). So for neuron \( j \), the weight vector will be:

\[ \mathbf{w}_j = [w_{j1}, w_{j2}, ..., w_{jm}]^T \]

For an input pattern \( \mathbf{x} \), compute \( \| \mathbf{w}_j - \mathbf{x} \| \) for each neuron, and choose the smallest value so obtained. Let \( i(\mathbf{x}) \) denote the index (that is the \( j \) value) of the winning neuron (which would also be the output of a trained SOM).²

² Some references indicate at this point that minimising \( \| \mathbf{w}_j - \mathbf{x} \| \) is equivalent to maximising \( \mathbf{w}_j \cdot \mathbf{x} \), but this is not the case unless \( \mathbf{w}_j \) and \( \mathbf{x} \) are normalised.

2. **Cooperative Process**

**Concept of Neighbourhood**

The winning neuron is to be the centre of a neighbourhood of cooperating neurons. Let \( d_{ij} \) denote the distance between neurons \( i \) and \( j \) (exact definition depends on dimensionality of SOM). We want an indicator \( h_{ji} \) to measure the "degree of membership" of neuron \( i \) in the neighbourhood centred on neuron \( j \). This indicator should take its maximum value when \( j=i \), and should decrease (to 0) as \( d_{ij} \) increases (to \( \infty \)). One suitable indicator function would be the Gaussian function:

\[ h_{ji} = \exp(-d_{ji}^2/(2\sigma^2)) \quad (9.4) \]

This indicator function \( h_{ji} \) has the additional property that it is independent of the location of the winning neuron \( i \). The parameter \( \sigma \) (sigma)³ is called the effective width of the neighbourhood.

³ If you are not familiar with the names and symbols for Greek letters like \( \sigma \), go to http://www.cse.unsw.edu.au/~billw/greek-symbols.html
Indicator functions for 1-D and 2-D SOMs

- 1-D SOM: \( d_j = | - j | \): neurons are numbered in "map order"
- To determine distance between neurons in a 2-D map, we need to have a function \( r(i) \) that maps neuron \( i \) to its coordinates on the (discrete) 2-D map. Then
  2-D SOM: \( d_j = \sqrt{r(i) - r(j)} \), where \( \| \| \) is the usual Euclidean norm.4
  This formula would also work for a 3-D SOM (though the norm would be the 3-D norm).

---

### 3. Adaptive Process

This is where we update the weights. Here is the rule:

\[
\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{h}_{n}(\mathbf{x} - \mathbf{w}(n))
\]

where

- \( \mathbf{x} \) is the input pattern.
- \( \eta(n) \) is the learning rate.
- \( \mathbf{h}_{n}(\mathbf{x}) \) is the neighbourhood function

- \( -\mathbf{w}(n) \) appears in the final factor as a kind of forgetting term - to stop the weights going off to infinity, each update includes a small degree of forgetting or decay of the previous value of the weight.

Learning rate should decrease gradually with time \( n \). A simple learning rate schedule:

\[
\eta(n) = \eta_0 \exp(-n/\tau_\eta),
\]

where

- \( \tau_\eta \) is the slope of the graph of \( \eta(n) \) against \( n \) - making \( \tau_\eta \) larger makes the learning rate \( \eta(n) \) decrease more slowly.

Both \( \tau_\eta \) and \( \tau_\eta \) are parameters for the algorithm - you try a few values until you get ones that give you convergence to sensible looking solutions in a reasonable amount of time.

---

The neighbourhood size shrinks with time

Another feature of the SOM algorithm is that the width of the neighbourhood shrinks with time:

\[
\alpha(n) = \sigma_0 \exp(-n/\tau_\alpha),
\]

where

- discrete step size \( n = 0, 1, 2, \ldots \)
- the constant \( \tau_\alpha \) determines the slope of the graph of \( \alpha(n) \) against \( n \) - making \( \tau_\alpha \) larger makes the width \( \alpha(n) \) decrease more slowly.

and the neighbourhood function \( h_j \) becomes \( h_j(n) \):

\[
h_j(n) = \exp\left(-\frac{d_j^2}{2\alpha(n)^2}\right) \tag{9.7}
\]

One effect of this is that as \( \alpha(n) \) approaches zero, only the winning node has a \( h \)-value that is significantly greater than zero. Since \( d_0 = 0 \), \( h_j(n) \) is always 1.

---

Two Phases of Adaptive Process

1. Self-organising or ordering phase …

- does topological ordering of weight vectors
- has maybe 1000 iterations
- starts with \( \eta(n) \) close to 0.1, decreases \( \eta \) gradually but keeps it > 0.01. E.g. \( \eta_0 = 0.1, \tau_\eta = 434 \)
- has a neighbourhood function \( h_{n\in}(\mathbf{x}) \) which initially includes almost all neurons (but centred on the winning neuron \( i \), but should shrink slowly with time until only a couple of neighbouring neurons figure significantly in the updates.

This could be achieved by setting \( \sigma_0 = \) the "radius" of the map, and \( \tau_\alpha = 1000/\log\sigma_0 \). Recall that \( \alpha(n) = \eta_0 \exp(-n/\tau_\alpha) \), so that \( \alpha(0) = \eta_0, \alpha(1000) = \eta_0 \exp(-\log\sigma_0) = \eta_0 \sigma_0^{-1} = 1.6

\[5\]

5 The radius of a 1-D map with \( n \) neurons is \( n/2 \). The radius of a \( m \times n \) 2-D map is \( (m+n)/2 \).

6 The SOM assignment instructions differ from this. Follow the instructions! The assignment also does not include a convergence phase.
Two Phases of Adaptive Process

2. Convergence phase
   • this phase fine tunes the feature map
   • the number of iterations \( \approx 500 \times (\# \text{ neurons in network}) \)
   • \( \eta(n) \) is maintained at a small value like 0.01
   • the neighbourhood function \( h_{ji}(n) \) should initially "contain" only a few neighbours, and decrease to only one or zero neighbours, as \( \sigma \) approaches 0.

Summary of SOM Algorithm

**Essential ingredients:**
- continuous input space of activation patterns that are generated in accordance with a particular probability distribution
- network in the form of a lattice of neurons defining a discrete output space
- time-varying neighbourhood function \( h_{ji}(n) \) defined around a winning neuron \( i(x) \)
- learning rate parameter \( \eta(n) \) that starts at \( \eta_0 \) and gradually decreases, but never reaches zero.

**Algorithm steps:**
1. **Initialisation:** choose small random values for the initial weight values \( w_j(0) \) - these vectors must be distinct. Alternatively, choose the initial weight vectors from the available set of input vectors \( \{x_i\}_{i=1,...,N} \). Either way, the dimension of the weight space = that of the input space.
2. **Sampling:** Choose a vector \( x(n) \) at random from the input space. Note that this is affected by the probability distribution mentioned above, so some regions of the input space will be more likely to contain the sample than others.
3. **Similarity Matching:** Find the winning neuron, that is, the neuron \( i(x) \) that best matches \( x(n) \), minimising \( \|x(n) - w_i\| \).
4. **Updating:** Adjust the synaptic weight vectors of all neurons by 
   \[ w_j(n+1) = w_j(n) + \eta(n)h_{ji}(n)(x - w_j(n)) \]
5. **Iteration:** repeat 2-4 until no noticeable map changes occur.

**Example A:** 2-dimensional input data forming 4 clusters \( \rightarrow \) 2-dimensional SOM
Example B: 2-dimensional input data → 2-dimensional SOM

Example C: 2-dimensional input data → 1-dimensional SOM

Properties of a SOM

- Let X be a continuous (input) vector space and A a discrete (but topologically ordered - e.g. 1-D or 2-D lattice) (output) space. What the SOM algorithm produces is a feature map \( \Phi: X \rightarrow A \) where \( \Phi(x) = i(x) \).
- Feature selection. Given an input space with a non-linear distribution, the SOM is able to select the set of best features for approximating the underlying distribution.
- Mapping a vector space into a discrete space is referred to as vector quantisation. Thus a SOM is a vector quantiser. A vector quantiser can be viewed as a classifier, or a data compression algorithm.

Data Compression

- Data compression algorithm come in two flavours: lossless and lossy.
- Data compression is done either to save storage space, or to optimise the use of the limited-capacity of a communication channel.
- Lossless compression algorithm make use of redundant information or inefficient encodings in order to achieve compression. The transformation is perfectly invertible (\( g = f^{-1} \)), so no information is lost when the data is decompressed. An example is the Huffman code algorithm.
- Lossy compression algorithms achieve compression by losing some of the precision in the data, so \( g \) is only approximately equal to \( f^{-1} \). Compression achieved using a SOM or other vector quantizer is of this type.
Vector Quantisation

- The input space is divided into a number of distinct regions, and for each region a reconstruction vector is defined.
- During encoding, a vector is represented by a compact label chosen by the quantizer and identifying the reconstruction vector for the input vector's region.
- During decoding, the labels are replaced by the reconstruction vectors.
- The collection of reconstruction vectors and corresponding labels is called the code book.
- For different choices of reconstruction vectors, the encoding-decoding sequence will cause a greater or lesser degree of distortion $D = 0.5 \int p(x) \delta(x, g(f(x)) \, dx$. Here $\delta(x, y)$ measures the difference between $x$ and $y$, and $p(x)$ is the probability of $x$.
- If we encode vectors with three 8-byte components (so 24 bytes) using, say 65,536 labels, (so 2 bytes), we would obtain 12-fold data compression. The cost is the loss of precision in the reconstructed vectors.

Two-Stage Algorithm for Finding Voronoi Vectors

The SOM algorithm provides approximate Voronoi vectors. It can be viewed as the first stage of a two stage process for solving this pattern classification task. The second (supervised) stage is learning vector quantization, which fine tunes the feature map so that the weight vectors correspond to the Voronoi vectors.

![Figure 9.13 Block diagram of adaptive pattern classification](image)

The LVQ Algorithm

The idea
Pick an input vector $x$ at random from the input space. If the class labels of $x$ and a Voronoi vector $w$ agree, move the Voronoi vector $w$ in the direction of $x$. If they disagree, move $w$ away from $x$.

The details
Let $(w_j | j = 1, ..., l)$ denote the set of wannabe Voronoi vectors, and let $\{x_i | i = 1, ..., N\}$ be the points in the input space. We assume $N >> 1$.

1. Choose an input vector $x_i$.
2. Suppose $w_j$ is the weight vector that is closest to input vector $x_i$. Let $C_{w_j}$ denote the class associated with $w_j$, and let $C_{x_i}$ denote the class label for $x_i$. Then $w_j$ is adjusted as follows:
   - If $C_{w_j} = C_{x_i}$ then $w_j(n+1) = w_j(n) + \alpha \left[ x_i - w_j(n) \right]$ towards $x_i$, where $0 < \alpha < 1$.
   - If $C_{w_j} \neq C_{x_i}$ then $w_j(n+1) = w_j(n) - \alpha \left[ x_i - w_j(n) \right]$ away from $x_i$.
3. The other Voronoi vectors are not modified.
4. The learning constant $\alpha$ should decrease monotonically with the number of iterations $n$. Initial value around 0.1, say.
5. After several passes through all the input data, the Voronoi vectors should converge.
Example of Adaptive Pattern Classification

Suppose we have two overlapping populations, as illustrated in panels (a) and (b) above. We wish to classify, based on position, which population a data point belongs to. Perfect classification is not possible, as in the region of overlap there is no certain way to know which class the point belongs to.

We can do a SOM on the input data, and then label each neuron in the map depending on how it responds to test data drawn from the input distribution. Then we do LVQ on the labelled neurons using the original test data.

Performance Improvement from LVQ

The approximate Voronoi vectors provided by the SOM algorithm could be used as a classification system in their own right. In 10 trials reported in Haykin, the SOM codewords led to average correct classification of 79.61% of 30,000 input patterns (using a 5x5 feature map).

The optimum classifier for this experiment will only get 81.51% right (because of the overlap of the two classes).

With LVQ applied to the feature map produced by SOM, the average percentage right increases to 80.52%.

This can be viewed either as an average improvement of 0.91% over SOM by itself, or you can look at the proportion of possible improvement provided by LVQ:

$$\text{actual improvement} = \frac{80.52 - 79.61}{1.90 - 79.61} = 0.91$$

So in this case, LVQ provided on average 48% of the improvement that is possible.

Applications of Kohonen's SOM

Try your favourite search engine. If this doesn't work for some reason, try http://www.google.com with the keywords Kohonen SOM applications.

A couple of examples to be going on with, though:

http://websom.hut.fi is about ordering search engine results so that similar documents are adjacent to each other (using SOMs).


http://www.cis.hut.fi/picsom is about a SOM-based image-browsing system.

Homework: check out these sites.
### Summary

<table>
<thead>
<tr>
<th>Self-organisation/Kohonen Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>• neural systems can learn without an explicit teacher;</td>
</tr>
<tr>
<td>• Kohonen's self-organizing map learns something about the structure of the input data space;</td>
</tr>
<tr>
<td>• SOM algorithm iterates over:</td>
</tr>
<tr>
<td>– competitive process (pick winning neuron);</td>
</tr>
<tr>
<td>– cooperative process (winner's neighbours win too);</td>
</tr>
<tr>
<td>– adaptive process (update weights);</td>
</tr>
<tr>
<td>• one application of SOMs is quantisation;</td>
</tr>
<tr>
<td>• The kernel SOM algorithm is a later, more complex version with improved performance in some situations.</td>
</tr>
</tbody>
</table>