Self-Organized (or Unsupervised) Networks.

The Delta-rule (including the Backpropagated version), and RNF networks, are all *supervised networks*. That is, we imagine that the data supplied to the net is split into two parts:

input from some (other) system (or from the environment), and *desired output*.

The task of the network has been to produce appropriate outputs for both the seen (training) and unseen (test) input data.

Another way of looking at some of these networks is to consider that the network *organises itself* as a result of the input (i.e. the *whole input* - both external input and externally supplied desired output).

This behaviour is clearly seen in the hidden units in a BP Delta rule network.

These units extract some useful information from the network input and form a *representation* of the input data

Unsupervised Networks.

Can we design networks whose *primary* function is to organize themselves in response to their input in such a way as to extract something *useful* from their input?

This presupposes there *is* something useful to be extracted. There are occasions when there is nothing to be extracted-

* purely randomly generated vectors, spaced throughout the input space
* data with no *redundancy*. (e.g. binary coded integers between 0 and 255, coded in 8 binary digits, where each digit is equally likely)

Most data (particularly data directly extracted from the environment, as opposed to data already coded by people) has a great deal of *redundancy*.

Redundancy is a relative term: it depends what one is trying to do. For example, a speech signal is highly redundant, if one is interested in transcribing it, or identifying the speaker; however, if one is considering the speech as sound for high-quality reproduction the signal is not redundant at all! Similarly, visual data is highly redundant if one is interested in navigating without colliding with objects, but not so redundant if one is concerned with precise identification of all the objects visible, and even less so if one is concerned with producing a precise copy.

What could be extracted from data with redundancy?

Prototyping
which prototypical pattern is the current pattern most like?

Clustering
if the data supplied falls into groups, which group does the current pattern belong to?

Redundancy reduction
forming a lower-dimensional (or less redundant) representation of the input pattern

Similarity
how similar is this pattern to previous ones?

Clearly, these possibilities are not distinct.

Self-organised feature extraction.

Networks which perform self-organised feature extraction attempt to train output units to respond to specific *features* in the input.

Usually the output units have an output value in the range [0..1], where 0 means *feature not present*, and 1 means *feature definitely present*.

We can consider this as as a fuzzy set-membership function: each output unit codes the degree to which the input is a member of some set, characterised by the presence of some feature in the data. 0 means *not a member of the set*, 1 means *definitely a member of the set*, and intermediate values have intermediate meanings.

These networks perform *data transformation* (perhaps even *data interpretation*).
Examples:

A network output unit might code
* the presence of a vertical black/white transition at some point in an image
* the presence of a letter *s* near the end of a string.

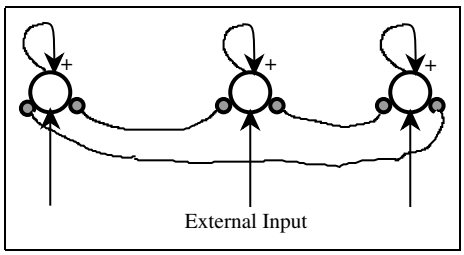
Competitive Learning

is a term used for networks which have a set of output units *competing* for activation. There are many ways in which this can be arranged: the simplest one is the *winner-take-all* cluster.

Winner-Take-All (WTA) clusters

A WTA cluster aims to make one output unit reach a maximum output, while the output of the others goes to a low level.

A winner-take-all cluster can consist of a number of units each exciting itself, and each inhibiting all the other units.



A 3-unit WTA cluster.

Shaded circles represent inhibitory connections. These connections implement *lateral inhibition*: their weights are usually fixed.

The effect is that the unit with the greatest activation *wins* and 'takes all the activation'.

Writing $Y_i(t)$ for the output of the unit at time t, and $A_i(t)$ for the activation of each unit at time t, we have

$$A_i(t+1) = I_i(t) + w_{ii}Y_i(t) + \sum_{j \neq i} w_{ij}Y_j(t)$$

followed by $Y_i(t+1) = F_{output}(A_i(t+1))$

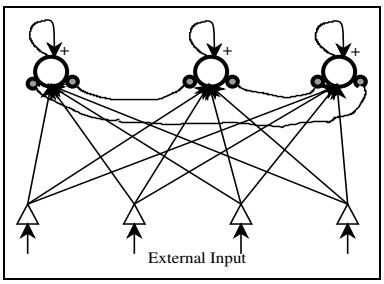
where $w_{ii} > 0$ (self-excitation) and $w_{ij} < 0$ ($i \neq j$) (lateral inhibition).

Given suitable weights, etc., the system will settle into a state where the Y_i corresponding to the highest I_i is at a much higher level than the others.

The system produced is a dynamic system. Such systems can oscillate, or produce other (relatively) unexpected behaviour. Here, I have described it using discrete time, but they can also be characterised using differential equations.

Note that the precise selection of the parameters can be quite difficult.

One can produce a *competitive learning system* by simply using an N-unit WTA cluster, with weights from each input unit.



Such a network is feed-forward between layers: it is, of course, recurrent as a whole.

If $\mathbf{W} = [w_{ij}]$, then for some unit i, $\Sigma_j w_{ij} I_j$ will be maximal, so that unit i will win.

But how can we set the **W** so that the system forms a useful classifier?

We could use a simple Hebbian rule:

$$\Delta w_{ij} = \eta I_j Y_i$$

but this, leads to weights growing without bound. Since the weight change is applied only to the winning unit, the effect is to make the winning unit more likely to win on other patterns too.

An alternative is to normalise the weights, so that $|\mathbf{W}| = 1$, or so that $\Sigma_j w_{ij} = 1$ for each unit i. Or we could use

$$w'_{i*j} = \eta I_j Y_{i*} + w_{i*j} \quad (1)$$

for the winning unit i* then scale the weights by

$$w''_{ij} = \alpha w'_{ij} \quad \text{for all i. } (\alpha < 1)$$

This reduces each weight after altering one of them, and has the advantage of locality (i.e. all the information required for weight modification is local to the units involved adjacent to the synapse).

This is very similar to using

$$\Delta w_{ij} = \eta[\frac{I_j}{\sum_k I_k} - w_{ij}]Y_i \quad (2)$$

In this case, the weights on the winning unit tend to $I_j/\Sigma_k I_k$, so that $\Sigma_j w_{ij} = 1$ and $\mathbf{W}_i = k\mathbf{I}$ so that $\mathbf{W}_i.\mathbf{I}$ is maximal.

Another possibility is to normalise the input first, then use

$$\Delta w_{ij} = \eta[I_j - w_{ij}]Y_i$$

moving the $\mathbf{W}_i$ directly towards the **I**.

Note: if we assume that the winning unit's $Y_i = 1$, and all the others have output 0, we can omit Y_i from the formulae.

In essence these are all post-synaptic rules: weight update is predicated on high activity in the post-synaptic unit.

If we normalise **I** and $\mathbf{W}_i$, then we can use a geometric analogy: the **I**'s are points on the surface of a (hyper)sphere, as are the $\mathbf{W}_i$'s: the effect of the learning rule is to move the $\mathbf{W}_i$'s towards the **I**'s.

Thus, if the **I**'s cluster, the $\mathbf{W}_i$'s will tend to move towards the centres of these clusters.

Note that (1) is always a local learning rule, and that (2) is local as well if we can assume that $\Sigma_k I_k = 1$.

There may be problems with *dead units*: that is, units which never win. Such units never have their weights updated: thus (continuing the geometric analogy), if their $\mathbf{W}_i$ vector pointed in a direction well away from any of the **I**'s at the start, they may never respond to any **I**.

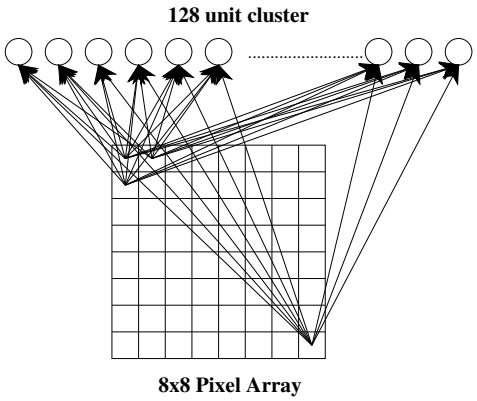
This may be a feature (if we expect to eventually receive totally different **I**'s at some point in the future), or a problem (we have units whose representation capacity is unused). Given access to the **I**'s initially, we can choose to initialise the $\mathbf{W}_i$'s so that the problem does not occur.

Vector Quantisation (VQ).

VQ is a method of ***using*** a competitive learning system, usually for data compression.

It is one of a class of *codebook* algorithms for this task: these are characterised by the replacement of some (large) elements of the data by codes from a much smaller alphabet. The codebook is transmitted first (or is simply agreed beforehand), and then the codes representing the data, rather than the data itself is transmitted.

Using a WTA cluster to compress image data



We can replace each 8 by 8 pixel array by one of 128 codes (1 per unit in the WTA cluster). Each can be coded by a 7 bit string.
We achieve a compression of 64/7 = 9.14. (assuming each pixel was on or off)

If each original pixel was 8 bits long, we transmit
(7/(8*64)) bits per pixel
= 0.0136 bits/pixel.

This would probably give a severely degraded picture - but possibly still recognisable.

Notes:

(1) Clearly, we do not have a *faithful* code: we cannot recreate the original data.
(2)For efficiency, we want all the codes to be used approximately equally often.

Technique:

```
Decide on number of codes.
Initialise weights
Train on some large sample of X's,
using a conscience algorithm, and
avoiding dead units.
Freeze the weights.
```

Before starting the transmission of the data itself, send the weights. Then, instead of sending the **X**'s, send a code describing which output unit is 'on'.

The output says only which cluster **X** belongs to. The output is made up from the weight vector of the cluster output unit, but this may be sufficient for acceptable transmission.