Kohonen Nets : Unsupervised Learning

MLPs and RBFs require 'supervisor': tells network error

A Kohonen network learns for itself : it is unsupervised

Neocognitron and ART are also unsupervised methods.

Self organising behaviour allows the network to discover significant features of the input data without a supervisor

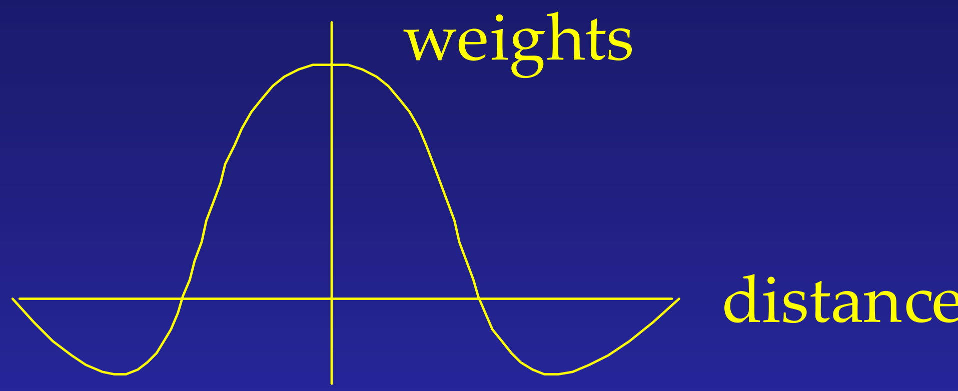
Idea: repeatedly present a set of input data & update network weights via SOM (Self Organising Map) training algorithm, until the net reaches some stable final configuration.

Often the SOM is 2 dimensional: its final configuration is a 2D topographic representation of the k-dimensional input data.

Hence, one application is data dimension reduction

Similar inputs to trained net excite similar regions of the SOM

Biological Justification



'mexican hat'

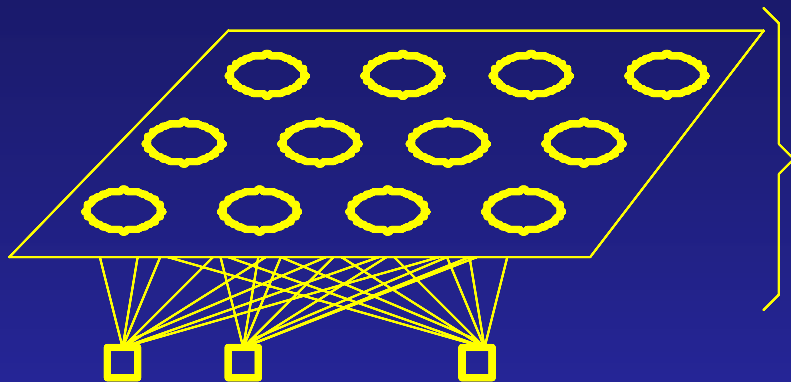
eg sinc(x)

Kohonen first developed scheme as a biologically plausible neural architecture modelled on the human cortex, which consists of a thin layer with rich lateral interconnections

- Interconnectivity strengths defined by 'Mexican Hat '
- Mexican Hat describes inhibitory and excitory connections between neurons as a function of separation distance.

But, normally computationally efficient networks are used.

Schematic Kohonen Network



Input N dimensions (normalised as calculating distances)

Each Neuron has N weights – initialise small random values

Input is connected to associated weight of ALL Neurons

Neurons normally as 2D rectangle (or hexagonally mapped)

Competitive learning used – node 'nearest' input wins!

Learning in a Kohonen Network

Initialise neuron weights to small random values.

Keep presenting inputs to the network;

For each presentation

Find the focus, *c: the node whose 'weights' are closest to input*

For all nodes i within Neighbourhood of *c*, N_c

$$W^{m}_{i}(t+1) = W^{m}_{i}(t) + \alpha(t)\,(X_i(t) - W^{m}_{i}(t))$$

Note learning rate $\alpha(t)$ should decrease with time

as should Neighbourhood N_c.

Finding focus … find Euclidean distance of data point and each node (ie n-dimensional Pythagoras on Data and W_i)

As want smallest distance, for efficiency, use distance squared

Topological Preservation

As network 'learns', nodes near each other 'learn' similar vals

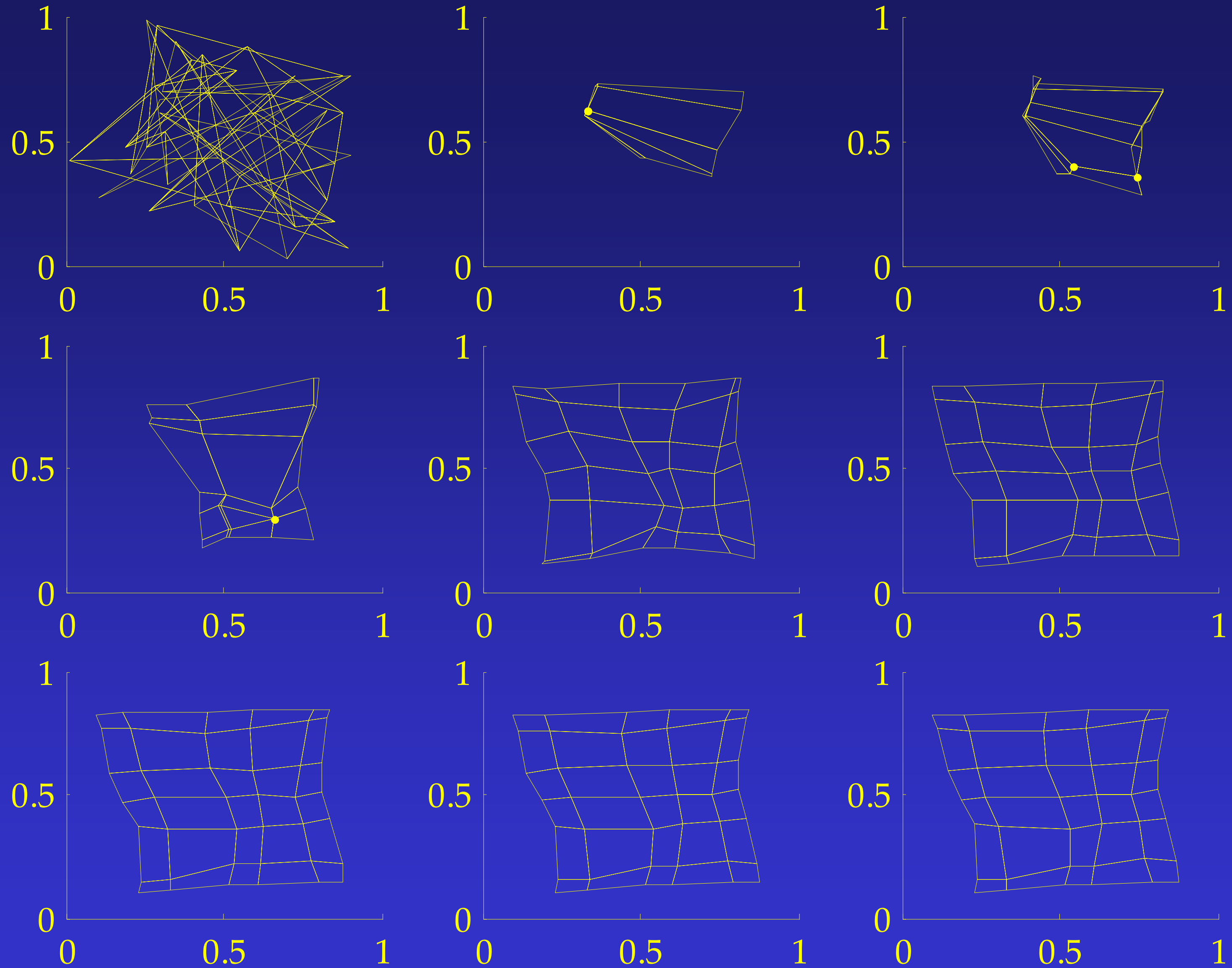
This is a consequence of the neighbourhood

This leads to the so-called 'topology preservation' property of Kohonen nets, which is demonstrated by having

- a 2D Kohonen network
- 2D data comprising random variables well spread over the range 0..1

Then as the network learns, so its weight vectors will '***self-organise***' into a regular lattice.

Next slide plot Weight 1 vs Weight 2 of each node, and draw lines between each node and its neighbours N S E W



Cybernetics

Function kn_learn

```
function net = kn_learn(net, ins, neigh, lrate, epochs);
% NET = KN_LEARN (NET, INS, NEIGH, LRATE, EPS);
% Do EPS times: Present each input in INS to network NET
% Find node nearest input and adjust it, and others in
    NEIGHbourhood, using learning rate LRATE
% NET is r*c*w array - r rows c cols and w weights; w vals in
    each row in INS
```

Called by following

```
>> net = rand (6,7,2);          % for 6*7 map, 2 inputs
>> ins = rand(200,2);           % 200 pairs of input values
>> neigh = [4 3 2 1 1 1 1 1];   % define shrinking neighbourhood
>> lrate = [0.5 0.4   …etc ] ;  % learning rate at each epoch
>> for ct=1:9 net = kn_learn(net, ins, neigh(ct), lrate(ct), 40); end
```

Function kn_output

kn_output finds relationship between input and each node

It returns, for each node,

either a) the distance squared between input and the node

or b) the sigmoid (on weighted sum of inputs) of the node

function netout = kn_output(net, ins, how);

% NETOUT = KN_OUTPUT (NET, INS, HOW)

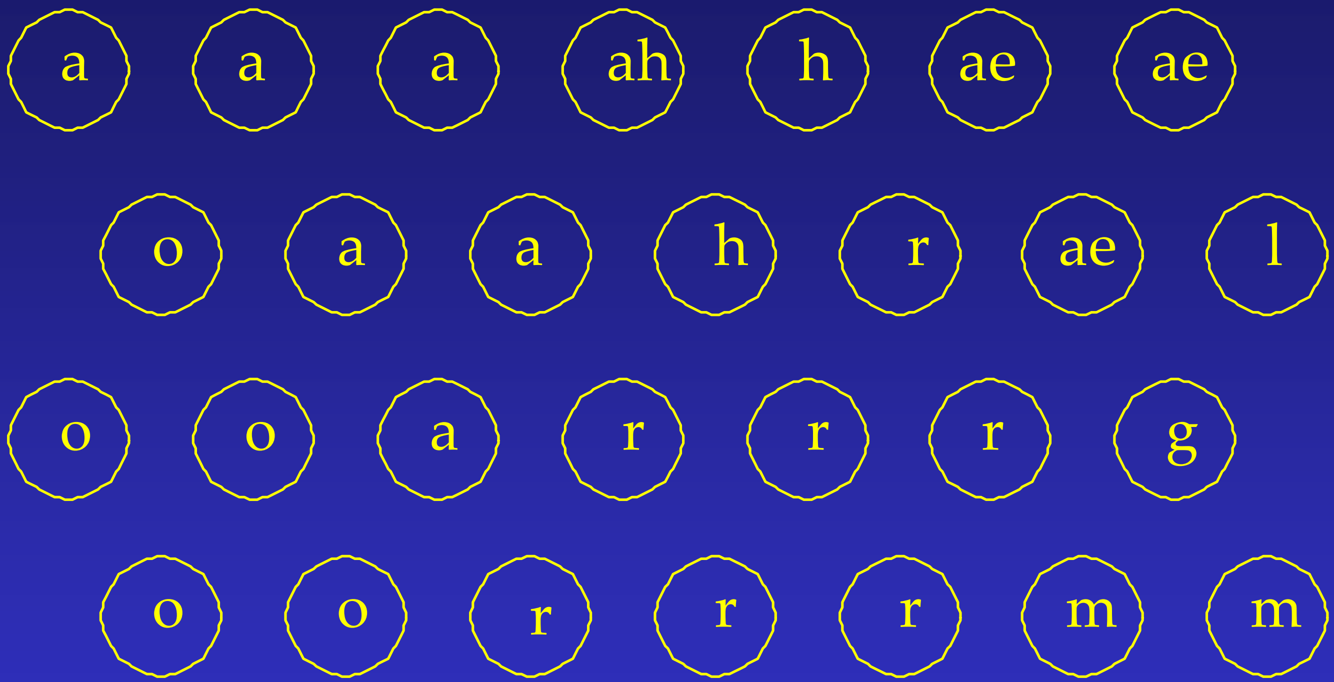
% HOW = 0, NetOut = Sigmoid(Weighted Sum of INS)

% HOW = 1, NetOut = Distance(Squared) from INS

Application - Speech Recognition

- Extract phoneme spectral data using FFT on input signal
- FFT is logarithmically filtered to produce a 15 element input vector. Typically in the range [200Hz .. 5KHz].
- Input vector elements are averaged and normalised.
- During learning each neuron begins to respond strongly to a specific phoneme.
- In use (speaker dependent 92-97% transcription accuracy):
  - At each sample period a specific neuron is most active.
  - If this is constant over a given time window, eg. 4 samples out of 7, then that phoneme is said to be active.
  - A word is identified as a recognisable trajectory over the feature map – moving from one phoneme to another

Part of (Hexagonal) Feature Map



Here are some nodes, and their phoneme labels.

Works very well in Finnish – but it is a phonetic language

Kohonen Maps & Dimension Reduction

A series of n-dimensional input vectors can be mapped, (classified), onto a say, two, dimensional feature map.

Network itself organises classification - no external direction.

Data could have 5 variables, 2D Map in effect has only 2!

Further reduction possible – by 'clustering'

eg. A Kohonen student classifier may have *n* different inputs corresponding to marks in particular subjects, amount of time spent working etc. and *m* output neurons.

After training it would be able to classify students into one of *m* groups!

Practical use – identifying gases in transformer & associating each region of SOM with particular fault or correct operation

Another Example

Example from Haykin. 16 animals

Dove, Hen, Duck, Goose, Owl, Hawk, Eagle, Fox

Dog, Wolf, Cat, Tiger, Lion, Horse, Zebra, Cow

For each there are 13 attributes (next slide) saying

whether is small, medium or big

has 2 legs, 4 legs, hair, hooves, mane of feathers

likes to hunt, run, fly, swim.

Each 'input' to network has these 13 attributes plus 16 numbers being all zeros except nth number of nth input is 0.2 (ie input specifies what the animal is.

Data are normalised so each vector has 'length' 1.

Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
Med	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2 leg	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4 leg	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Experiment

A Kohonen Network of 10*10*29 is initialised

The network is trained for 2000 epochs. Two outputs found

First Output is where take each item in data set and find the one node which is closest to the input and that node is labelled according to animal associated with that input.

Second, do ‘simulated electrode penetration’

Here take each node in the network and find which input is closest to it, and then label that node according to the associated input

The result is a ‘contextual map’

You will note that similar animals are adjacent …

First Table … Shows 'Winners'

			Fox		Dog				Tiger
Cat							Wolf		
					Eagle				Lion
Hen									
			Dove			clash			Horse
Goose									
		Duck			Cow				Zebra

Contextual Map ….

Cat	Cat	Fox	Fox	Dog	Dog	Wolf	Wolf	Lion	Tiger
Cat	Cat	Fox	Fox	Dog	Dog	Wolf	Wolf	Lion	Tiger
Cat	Cat	Cat	Eagle	Eagle	Eagle	Wolf	Wolf	Lion	Lion
Cat	Cat	Cat	Eagle	Eagle	Eagle	Eagle	Lion	Lion	Lion
Hen	Hen	Hen	Eagle	Eagle	Eagle	Eagle	Hawk	Lion	Lion
Hen	Hen	Dove	Dove	Dove	Hawk	Hawk	Hawk	Horse	Horse
Hen	Hen	Dove	Dove	Dove	Hawk	Hawk	Hawk	Horse	Horse
Goose	Goose	Dove	Dove	Dove	Hawk	Hawk	Hawk	Horse	Horse
Goose	Goose	Duck	Duck	Cow	Cow	Cow	Zebra	Zebra	Zebra
Goose	Goose	Duck	Duck	Cow	Cow	Cow	Cow	Zebra	Zebra

Note, for inst, Zebra near Horse …. Lion near Tiger

Application Determining Input Variables

Kohonen maps can be used to give an indication of whether different variables affect an output variable.

This can be useful in deciding which variables to analyse when forecasting data, for instance.

Here all variables (including output) are fed to network.

The network is trained as usual

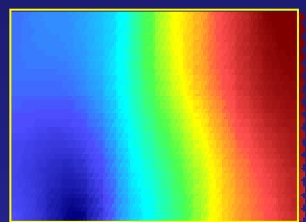
Then each plane is shown suitable colour coded

- Plane (n) is the value of the nth weight of each neuron
- Weights are in range 0..1; colours are mapped to this range.

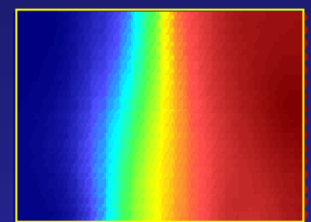
Look at planes for 'output' and each input: if have similar areas (normally diff. colours), worth using that input

Example Electricity Demand

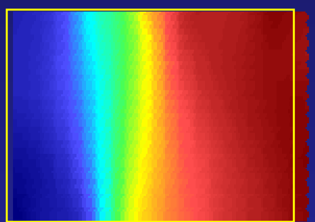
Temp_std



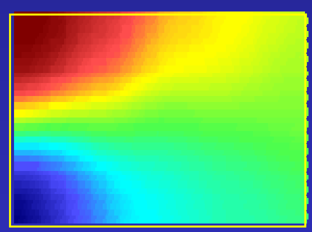
Illum_std



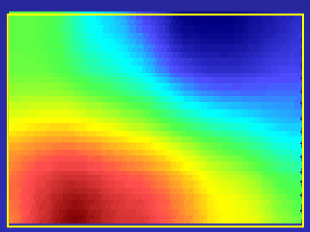
Demand_std



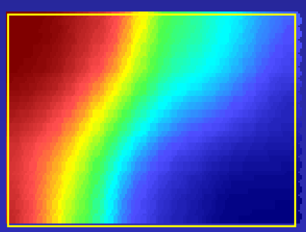
Time_std



Sin(Time*2π/24)



Cos(Time*2π/24)



Looks like temperature, illumination and (to lesser extent) cos(timeofday) are best indicators of demand

NB _std at end of variable names indicates each has been normalised using its std.