DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Lecture 6: **Self-organising maps (SOMs)**

Pawel Herman

Computational Science and Technology (CST)

KTH Royal Institute of Technology

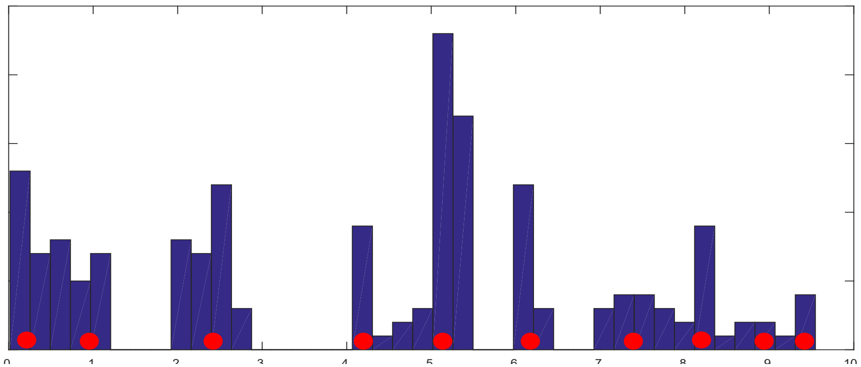
February 2018

Lecture overview

- Vector Quantisation
- Self-organising maps (SOMs)
- Learning vector quantisation

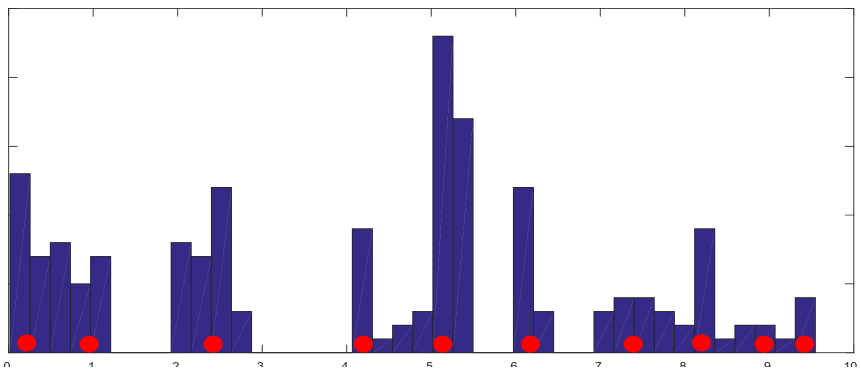
Vector Quantisation

- The idea borrowed from information coding, data compression
- Represent data in terms of few (limited) typical data vectors – *code vectors*
  - compression
  - noise reduction

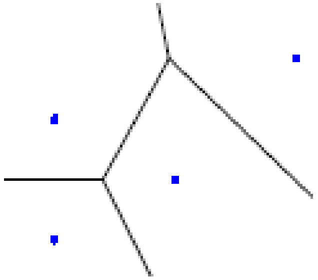


Vector Quantisation

- The idea borrowed from information coding, data compression
- Represent data in terms of few (limited) typical data vectors – *code vectors*
  - compression
  - noise reduction

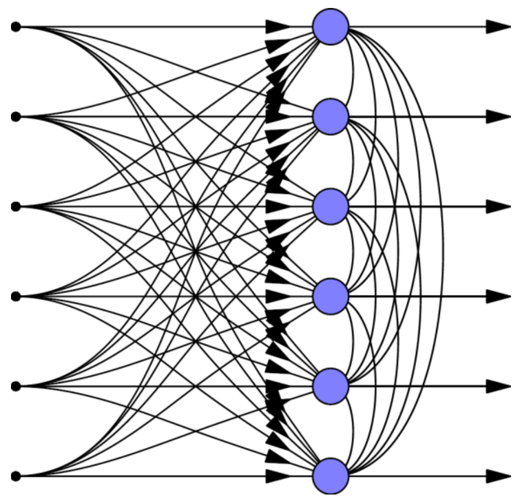


- Voronoi partitioning



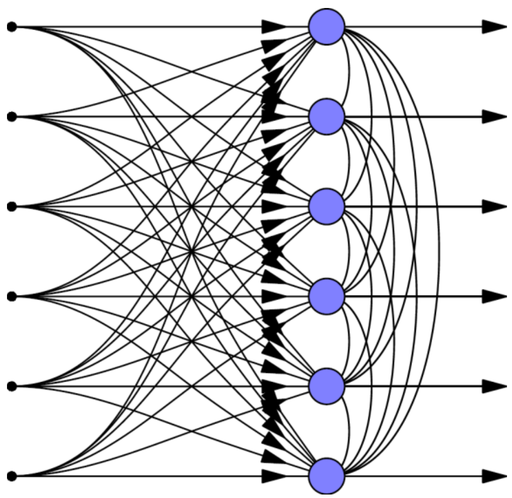
Vector Quantisation – unsupervised learning

- Network architecture for unsupervised learning



Vector Quantisation – unsupervised learning

- Network architecture for unsupervised learning



- Winner-take-all (WTA) mechanism
  - ONLY ONE output (the winner) unit gets active – hard competition
  - competition is one of the most fundamental computation in the brain via so-called *lateral inhibition*

Competitive learning – basic principle

The fundamental principle

Update the *winning* unit (prototype/code vector) to make it more specialised ("even better").

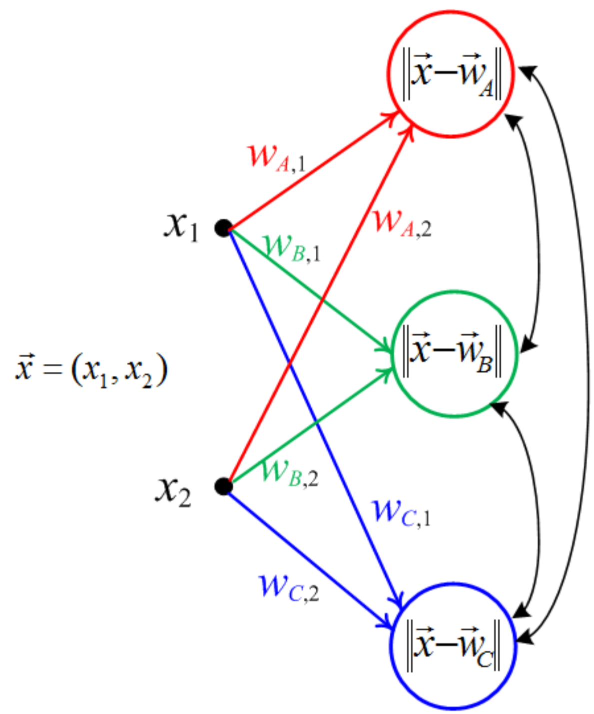
Competitive learning – basic principle

The fundamental principle

Update the *winning* unit (prototype/code vector) to make it more specialised ("even better").

Basic algorithm

- the normalised weight vectors, $\mathbf{w}_A$, …, $\mathbf{w}_C$ are generated randomly
- for each input vector, x, calculate "proximity" to weight vectors and determine the nearest neighbour
- make an update of the nearest weight vector and normalize weights



If the **red** node $\boldsymbol{w_A}$ wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

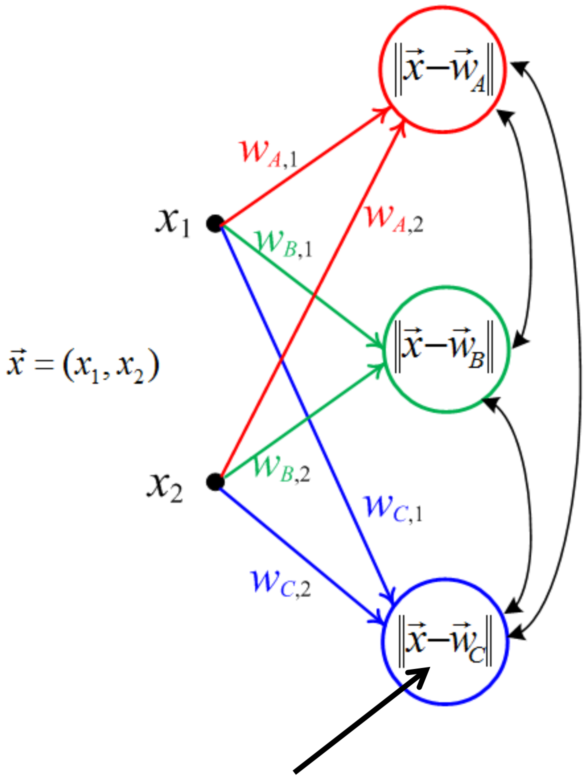
Competitive learning – basic principle

The fundamental principle

Update the *winning* unit (prototype/code vector) to make it more specialised (“even better”).

Basic algorithm

- the normalised weight vectors, $\mathbf{w}_A, \ldots, \mathbf{w}_C$ are generated randomly
- for each input vector, **x**, calculate **“proximity”** to weight vectors and determine the nearest neighbour
- make an update of the nearest weight vector and normalize weights



If the **red** node ***w**$_A$* wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

proximity definition determines how we find the winner (*scalar product* is more popular)

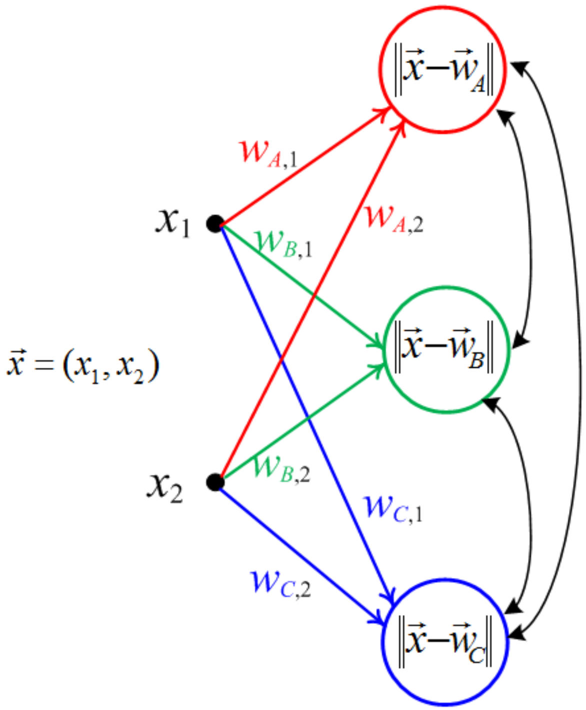
Competitive learning – basic principle

The fundamental principle

Update the *winning* unit (prototype/code vector) to make it more specialised ("even better").

Properties

- the algorithm finds cluster in data – purely unsupervised approach
- each node protects its "territory"
- there is also a batch version



If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

Competitive learning – problem with dead units

Dead unit problem

Prototype vectors far from actual data will never become better.

Competitive learning – problem with dead units

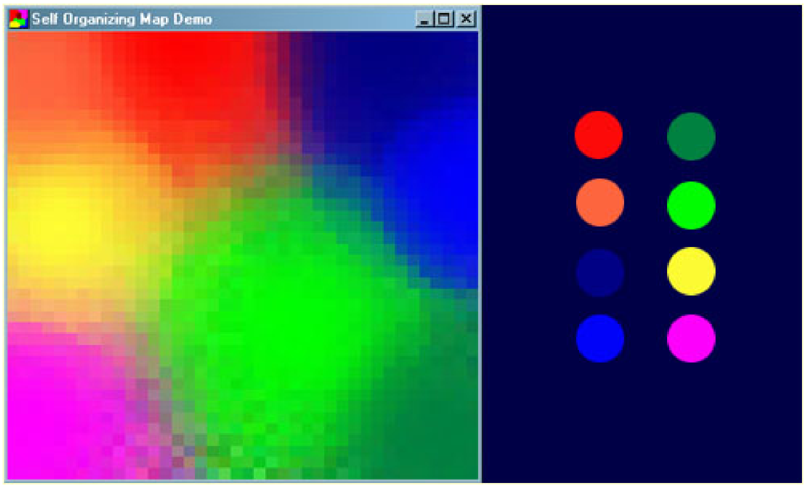
Dead unit problem

Prototype vectors far from actual data will never become better.

Methods to avoid dead units

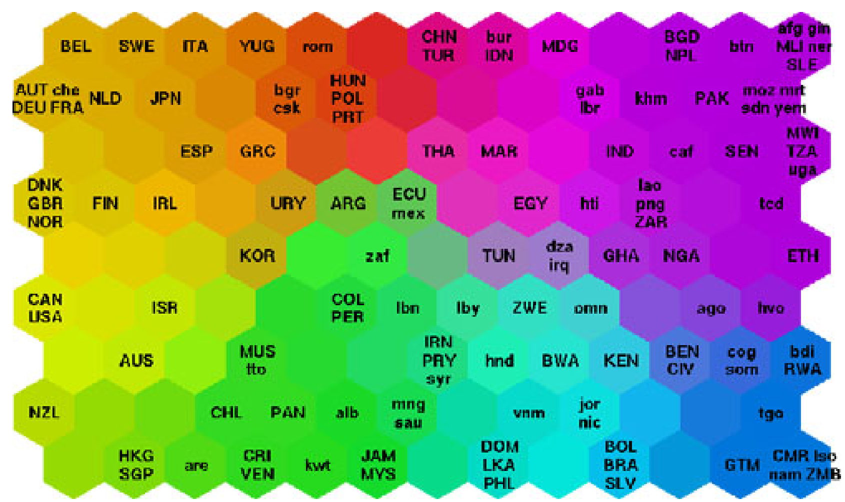
- Initialise algorithm (the weights) with data samples
- "Leaky learning" (soft-competition) – some updates for all
- Learning with conscience – balanced update allowing losers to win
- Introduce noise to data

Topographic map

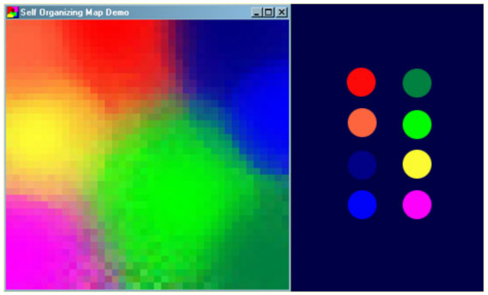


colours

poverty

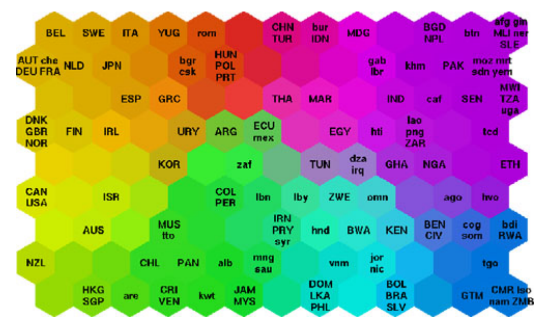


Topographic map



"The spatial location of an output neuron in a topographic map corresponds to a particular domain or feature drawn from the input space"

Bullinaria, 2004



Topology preserving maps

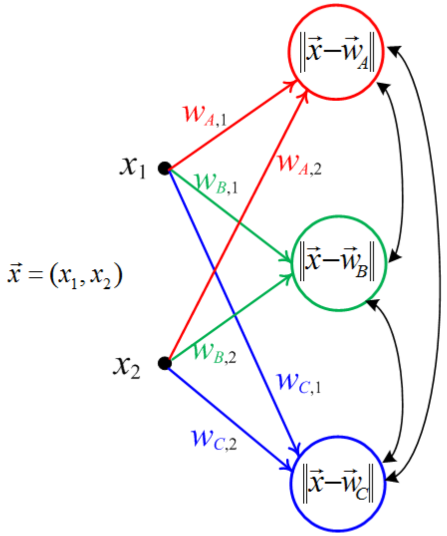
Learning principle

Competitive learning where winning “spills over” to neighbours.

Topology preserving maps

Learning principle

Competitive learning where winning "spills over" to neighbours.



Competitive learning

If the **red** node $\boldsymbol{w_A}$ wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

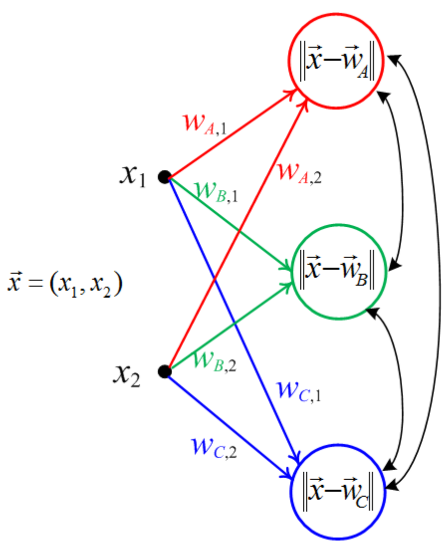
OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

Topology preserving maps

Learning principle

Competitive learning where winning "spills over" to neighbours.



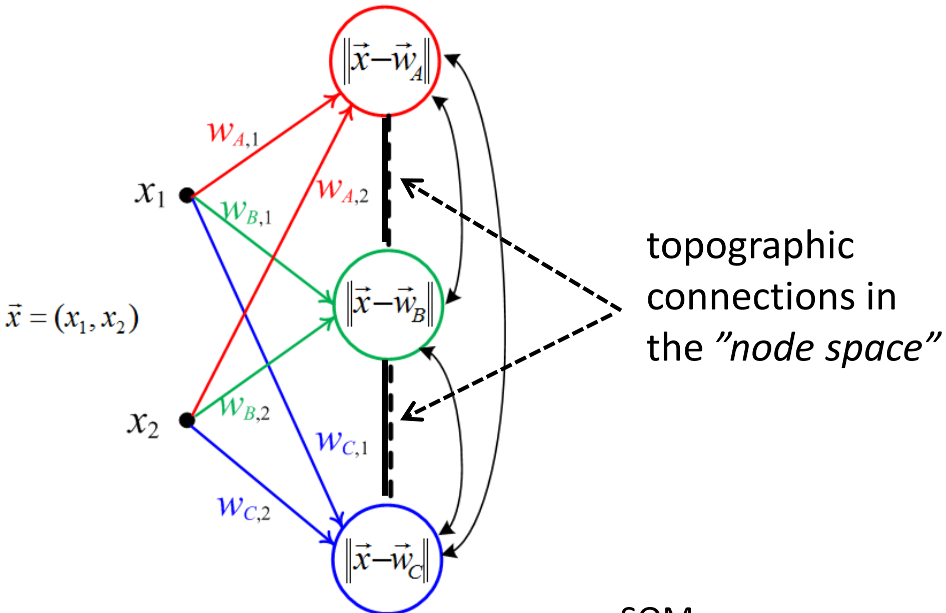
If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x}-\vec{w}_A)$$

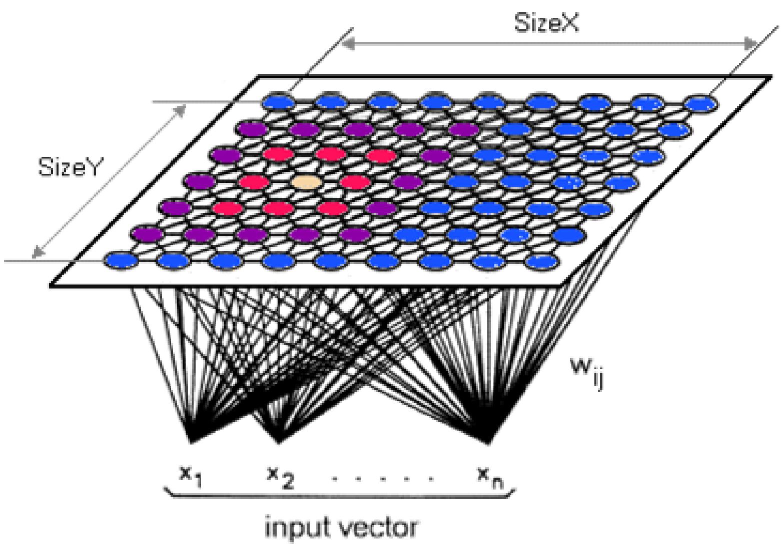
Competitive learning



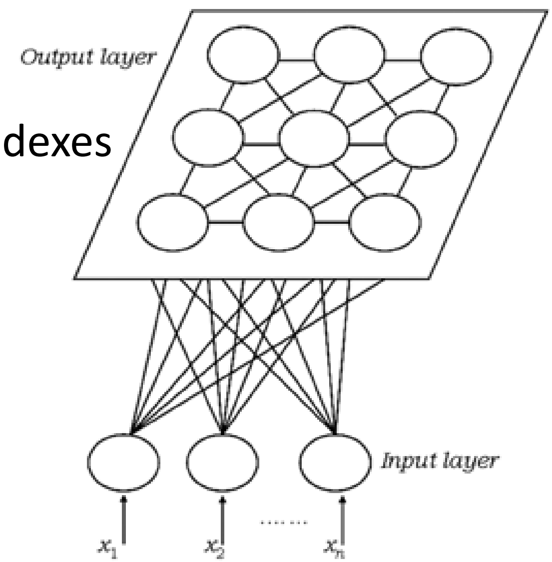
SOM

Mapping from high- to low-dimensional spaces

Low-dimensional (discrete) OUTPUT space (n<=2),
where output **nodes** are represented.



indexes

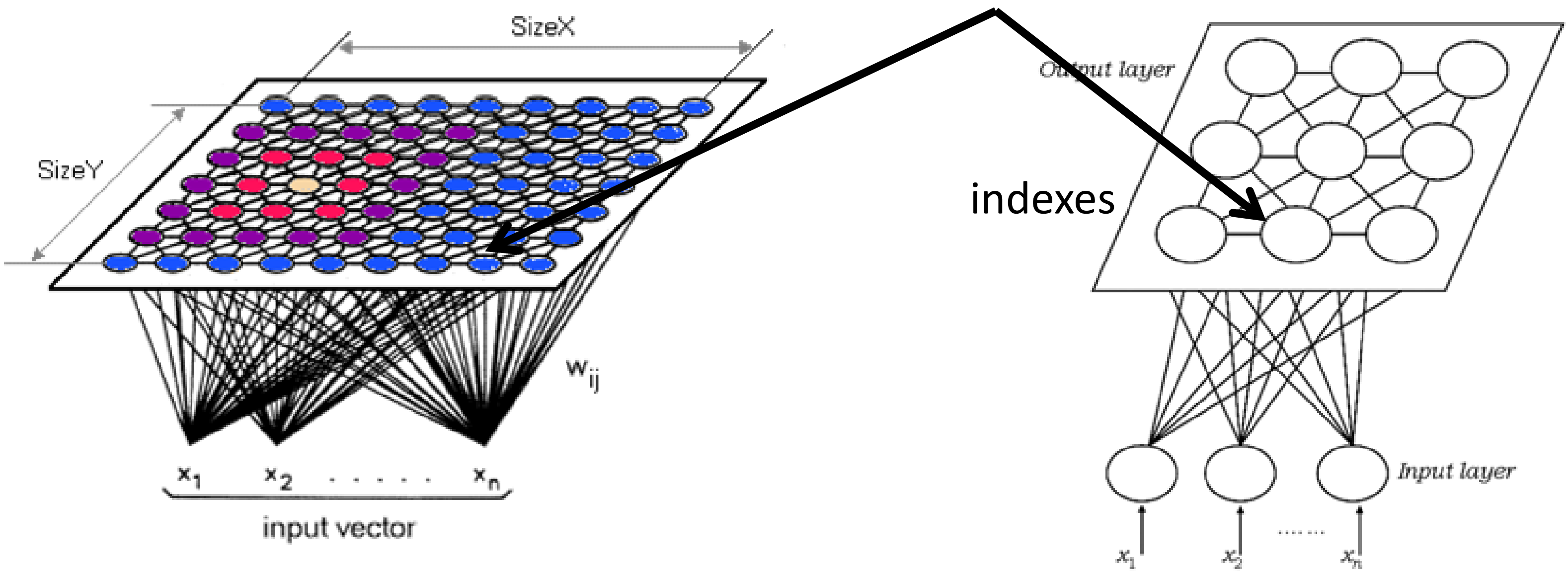


Multi-dimensional (often continuous) INPUT space (n>2),
where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

Mapping from high- to low-dimensional spaces

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined to determine.....

....which nodes apart from the *winner* (**best matching unit**) get their weights, ***w***, updated

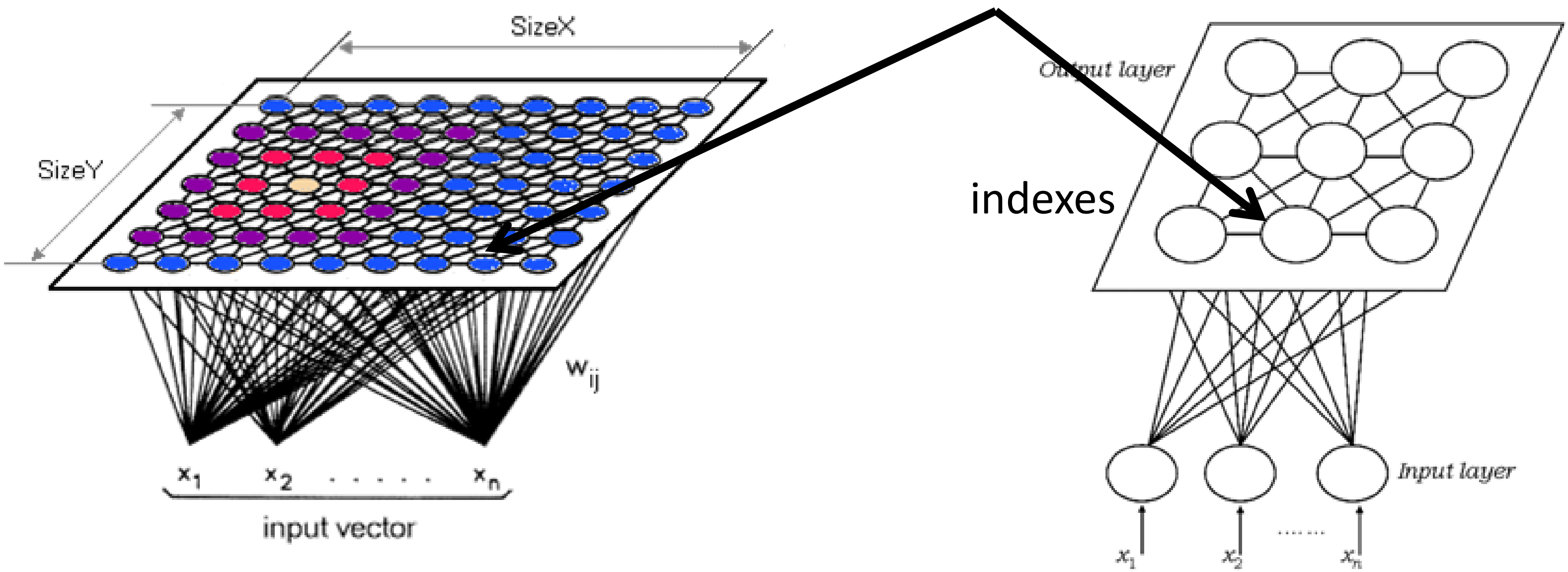


Multi-dimensional (often continuous) INPUT space (n>2), where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

Mapping from high- to low-dimensional spaces

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space

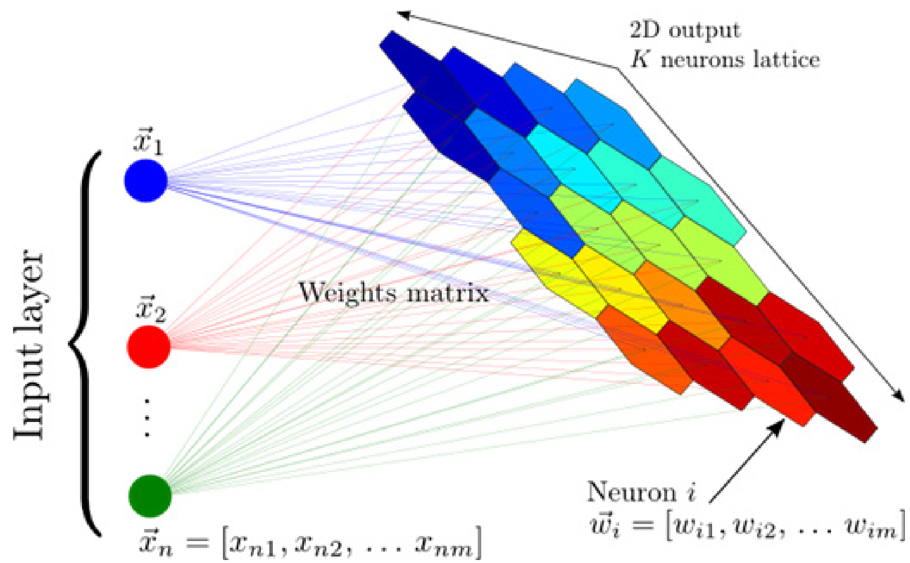


Multi-dimensional (often continuous) INPUT space (n>2), where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

Mapping from high- to low-dimensional spaces

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space



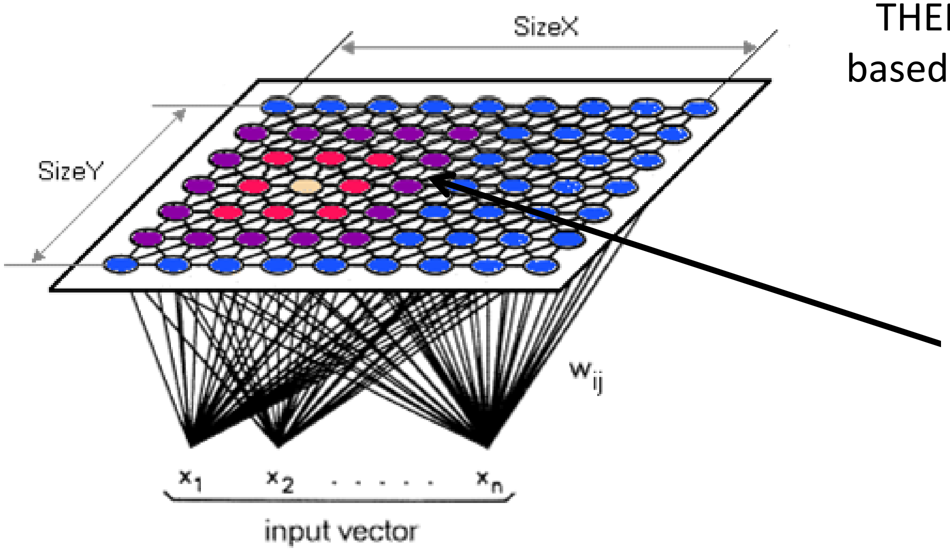
The links are not always shown explicitly

Multi-dimensional (often continuous) INPUT space (n>2), where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

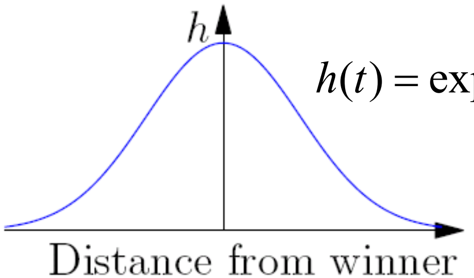
Distance and neighbourhood in the *output* space

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space



THEN we can use a neighbourhood function, *h*, based on the distance between the discrete nodes in the OUTPUT space



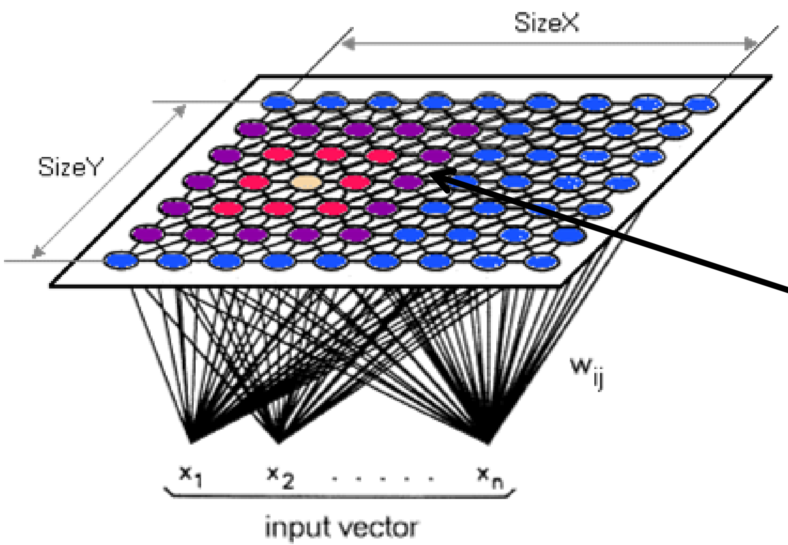
$$h(t) = \exp\left(-\frac{d^2}{2\sigma^2(t)}\right)$$

Multi-dimensional (often continuous) INPUT space (*n*>2), where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

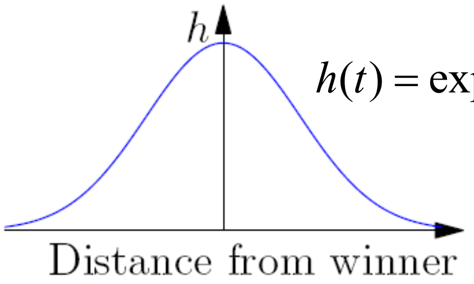
Distance and neighbourhood in the *output* space

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

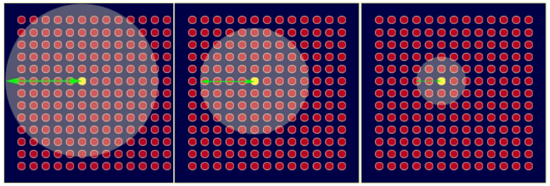
The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space



THEN we can use a neighbourhood function, *h*, based on the distance between the discrete nodes in the OUTPUT space



$$h(t) = \exp\left(-\frac{d^2}{2\sigma^2(t)}\right)$$

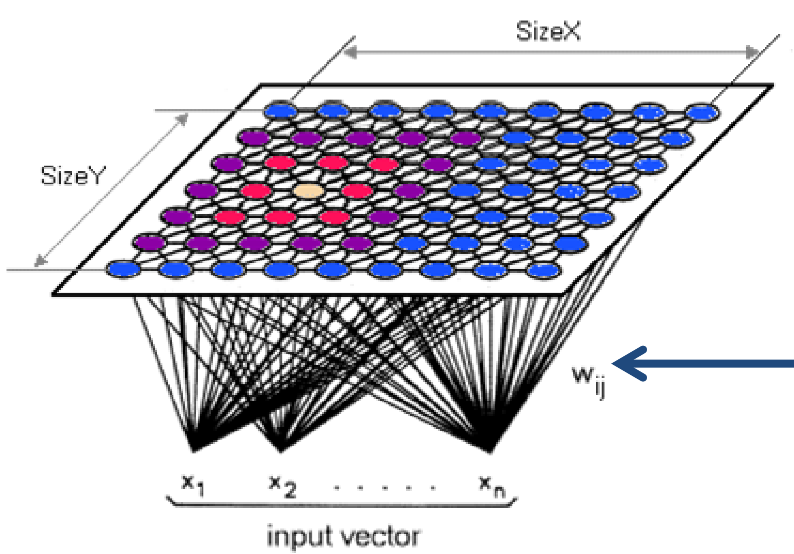


h shrinks since *σ* exponentially decreases over time: $\sigma(t) = \sigma_0 \exp\left(-\frac{t^2}{\tau}\right)$

Update of the corresponding *input* weights

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space



THEN we can use a neighbourhood function, *h*, based on the distance between the discrete nodes in the OUTPUT space

Finally, however, the weight vectors, ***w***, get updated (moved in the multi-dimensional input/weight space)

$$\Delta \boldsymbol{w} = \eta h(\boldsymbol{x} - \boldsymbol{w})$$

Multi-dimensional (often continuous) INPUT space (*n*>2), where both **inputs, *x*,** and **weight** vectors, ***w***, are represented.

SOM visualisation – showing lattices

In cases when the dimensionality of the INPUT space is low ($n<=3$)

SOMs are often visualized only in the INPUT/WEIGHT space

SOM visualisation – showing lattices

In cases when the dimensionality of the INPUT space is low (n<=3)

SOMs are often visualized only in the INPUT/WEIGHT space

and then....

... it is very useful to **show links between nodes** as they illustrate neighbourhood (topographical relationship) in the OUTPUT space

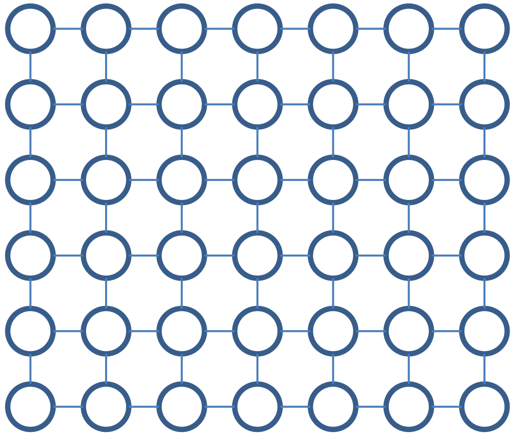
SOM visualisation – showing lattices in *input* space

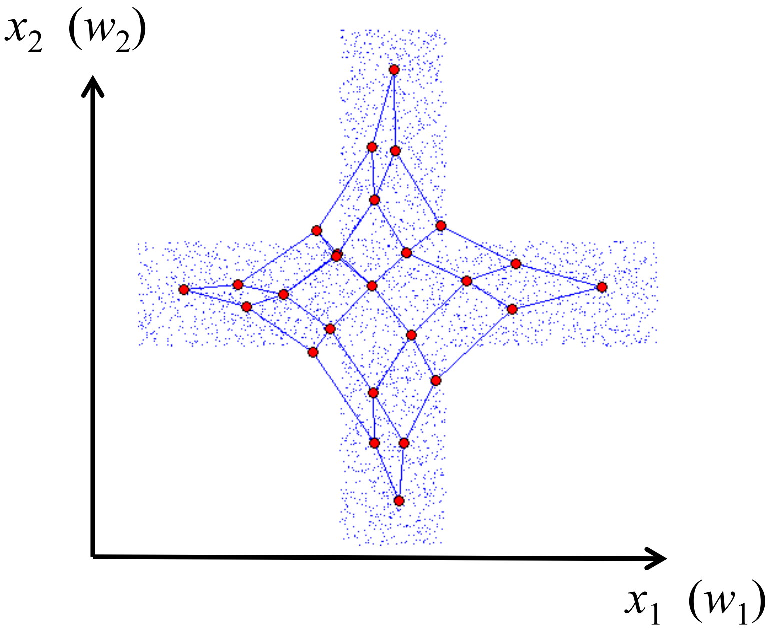
... it is very useful to **show links between nodes** as they illustrate neighbourhood (topographical relationship) in the OUTPUT space

---

2 inputs – **2D** INPUT space (x_1,x_2) corresponding to **2D** WEIGHT space (w_1,w_2)

2D arrangement of nodes (lattice) in the OUTPUT space





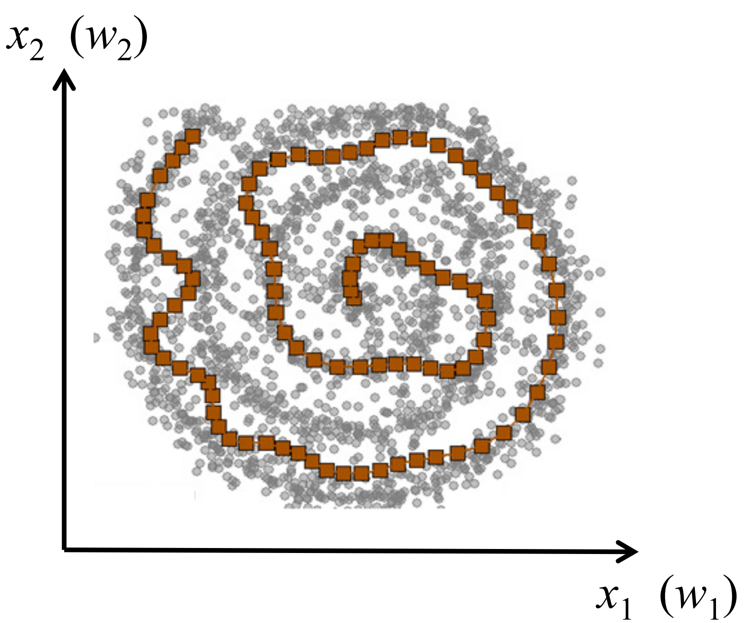
SOM visualisation – showing lattices in *input* space

... it is very useful to **show links between nodes** as they illustrate neighbourhood (topographical relationship) in the OUTPUT space

---

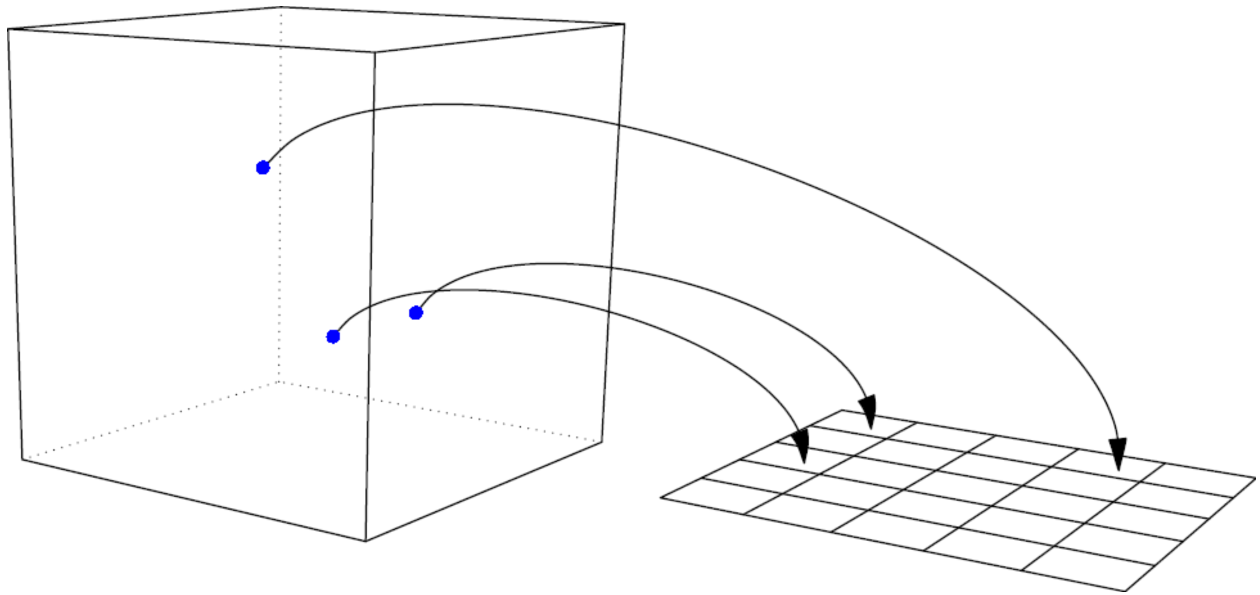
2 inputs – **2D** INPUT space (x_1,x_2) corresponding to **2D** WEIGHT space (w_1,w_2)

1D arrangement of nodes
in the OUTPUT space



SOM visualisation demo

Lower-dimensional manifold



SOM learning

Following initialisation there are three learning stages

1. *Competition* (mapping of a continuous input space onto a discrete output space of units)
2. *Cooperation* (lateral interaction through topographic neighbourhood)
3. Weight (synaptic) *adaptation*

SOM learning

Following initialisation there are three learning stages

1. *Competition* (mapping of a continuous input space onto a discrete output space of units)

2. *Cooperation* (lateral interaction through topographic neighbourhood)

3. Weight (synaptic) *adaptation*

   a) topological *ordering:* high learning rate and large neighbourhood

   b) *convergence:* low learning rate and small neighbourhood
   (a long process of fine tuning)

SOM learning

Following initialisation there are three learning stages

1. *Competition* (mapping of a continuous input space onto a discrete output space of units)

2. *Cooperation* (lateral interaction through topographic neighbourhood)

3. Weight (synaptic) *adaptation*

   a) topological *ordering:* high learning rate and large neighbourhood

   b) *convergence:* low learning rate and small neighbourhood

the key role of neighbourhood!

GLOBAL ordering vs LOCAL fit

SOM learning

Following initialisation there are three learning stages

1. *Competition* (mapping of a continuous input space onto a discrete output space of units)

2. *Cooperation* (lateral interaction through topographic neighbourhood)

3. Weight (synaptic) *adaptation*

   a) topological *ordering:* high learning rate and large neighbourhood

   b) convergence: low learning rate and small neighbourhood

the key role of neighbourhood!

"Global order can arise from local interactions"
Turing

GLOBAL ordering vs LOCAL fit

SOM learning

Following initialisation there are three learning stages

output space

1. *Competition* (mapping of a continuous input space onto a discrete output space of units)
2. *Cooperation* (lateral interaction through topographic neighbourhood)

input space

3. Weight (synaptic) *adaptation*
   a) topological *ordering:* high learning rate and large neighbourhood
   b) convergence: low learning rate and small neighbourhood

SOM example

High-dimensional feature description

	Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
Large	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Fur	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Hoofs	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
Hunter	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
Runner	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
Flyer	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Swimmer	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

SOM example

2D visualisation

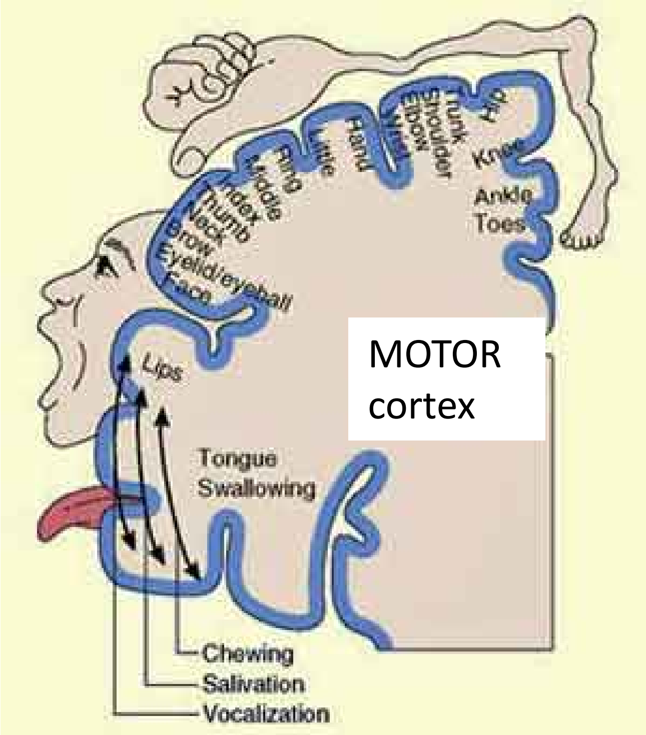
Dog	.	.	Fox	.	.	Cat	.	.	Eagle
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	Owl
.	.	.	.	.	.	Tiger	.	.	.
Wolf	.	.	.	.	.	.	.	.	Hawk
.	.	.	Lion	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	Dove
Horse	.	.	.	.	.	.	Hen	.	.
.	.	.	.	Cow	.	.	.	.	Goose
Zebra	.	.	.	.	.	.	Duck	.	.

SOM example

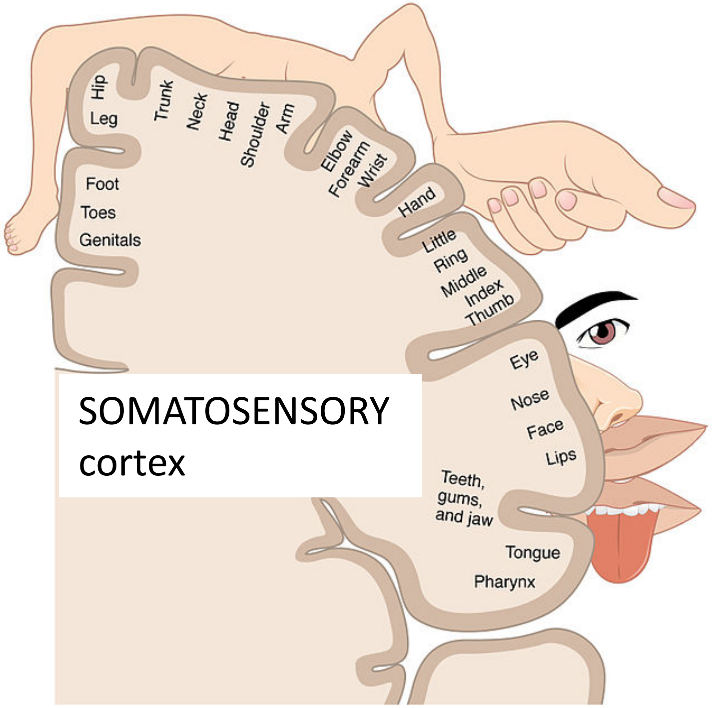
2D visualisation

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

The brain analogy of topographic mapping

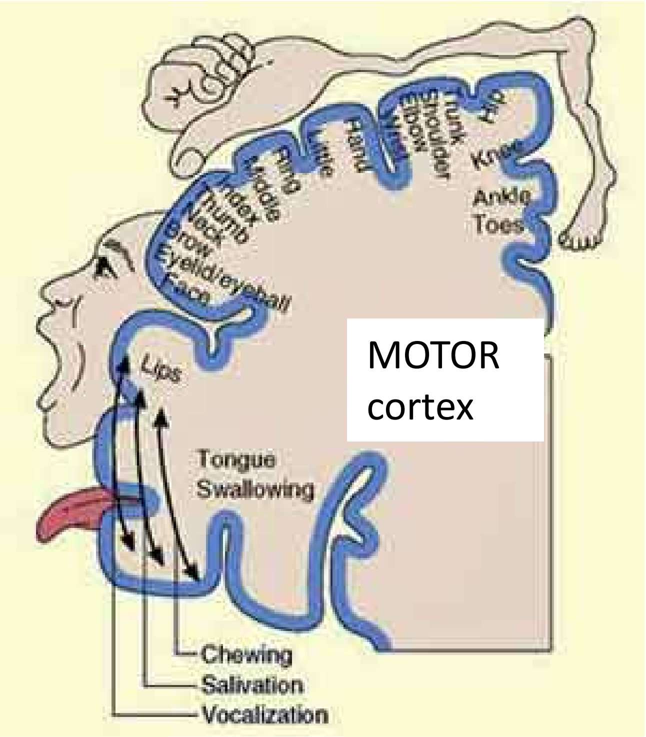


"Kohonen map captures the essential features of computational maps in the brain"

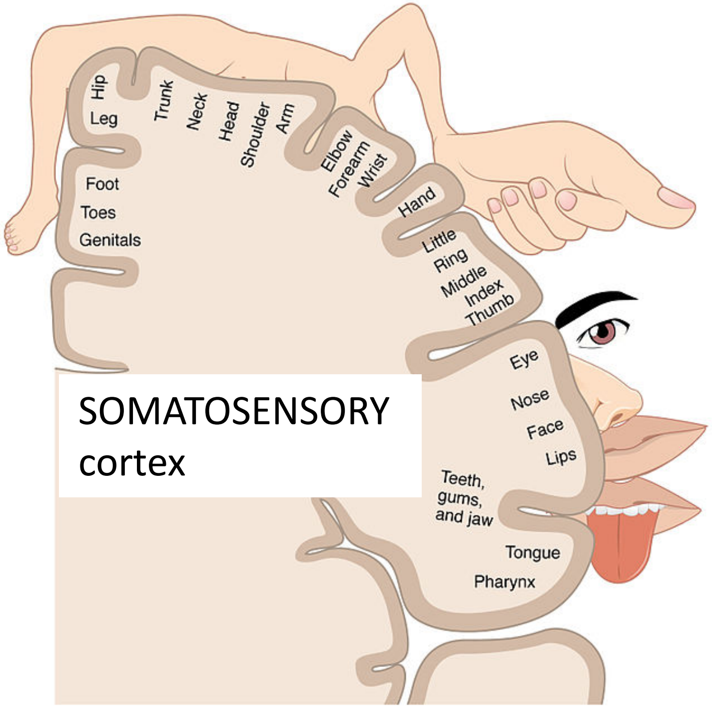


Sensory experience is multi-dimensional, so there is a need to map them to spatial relations in the cortex preserving topology.

The brain analogy of topographic mapping



"Kohonen map captures the essential features of computational maps in the brain"



Sensory experience is needed for developing neural circuitry through analogous *neurobiological processes*.

Learning Vector Quanitsation (LVQ)

Learning Vector Quantisation (LVQ) is a *supervised* competitive learning algorithm (classes are known)

Learning Vector Quanitsation (LVQ)

Learning Vector Quantisation (LVQ) is a *supervised* competitive learning algorithm (classes are known)

$$\Delta\vec{w} = +\eta(\vec{x} - \vec{w})$$

if the winner belongs to the *right* class

$$\Delta\vec{w} = -\eta(\vec{x} - \vec{w})$$

if the winner belongs to the *wrong* class