EE5904/ME5404:
Lecture Six
Self-Organizing Maps

Xiang Cheng

Associate Professor

Department of Electrical & Computer Engineering
The National University of Singapore

Phone: 65166210 Office: Block E4-08-07
Email: elexc@nus.edu.sg

Learning

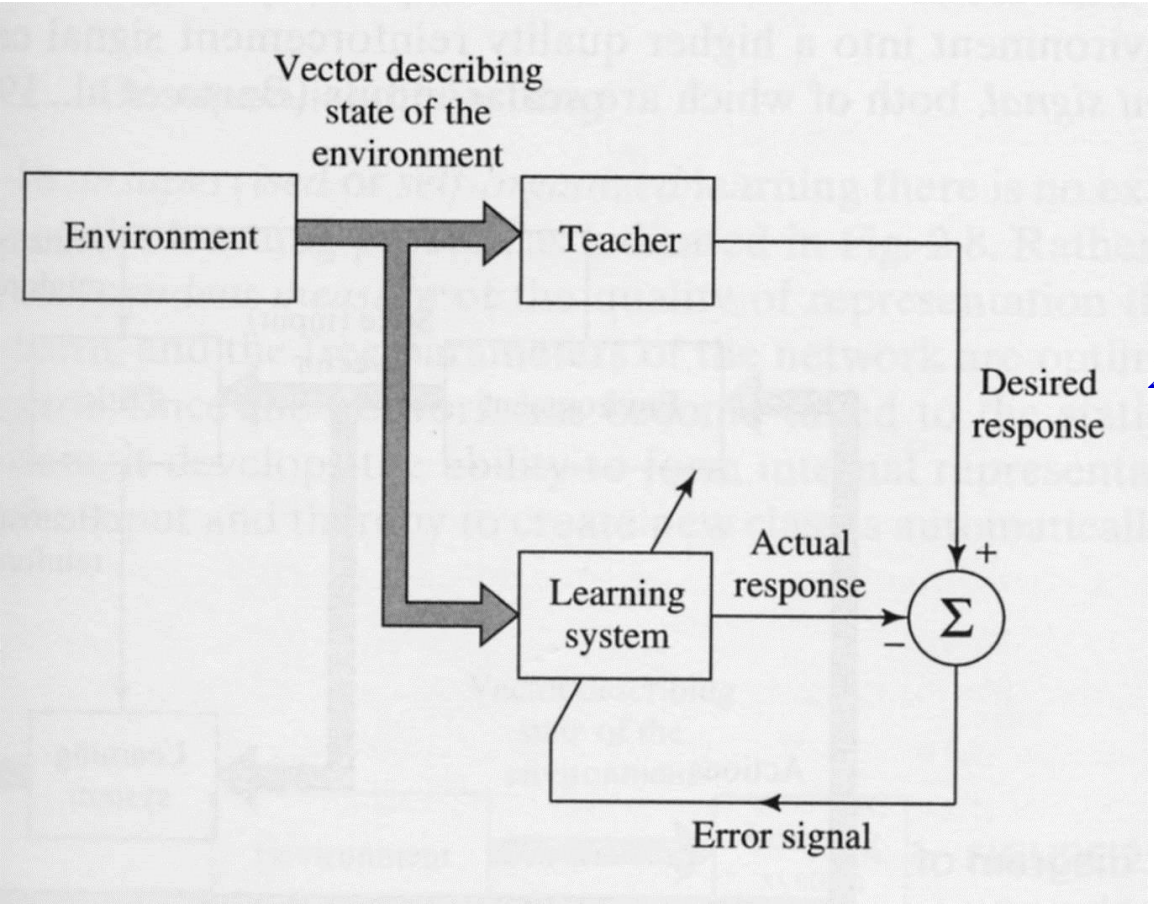
What is learning in neural networks?

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

Process of learning:

1. *The neural network is stimulated by an environment*
2. *The neural network undergoes changes in its free parameters as a result of this stimulation.*
3. *The neural network responds in a new way to the environment because of the changes that have occurred in its internal structure.*

Learning with a teacher: Supervised Learning



Process of learning:

1. *The neural network is fed with input, and produce an output.*

2. *The teacher will tell what the desired output should be, and the error signal is generated.*

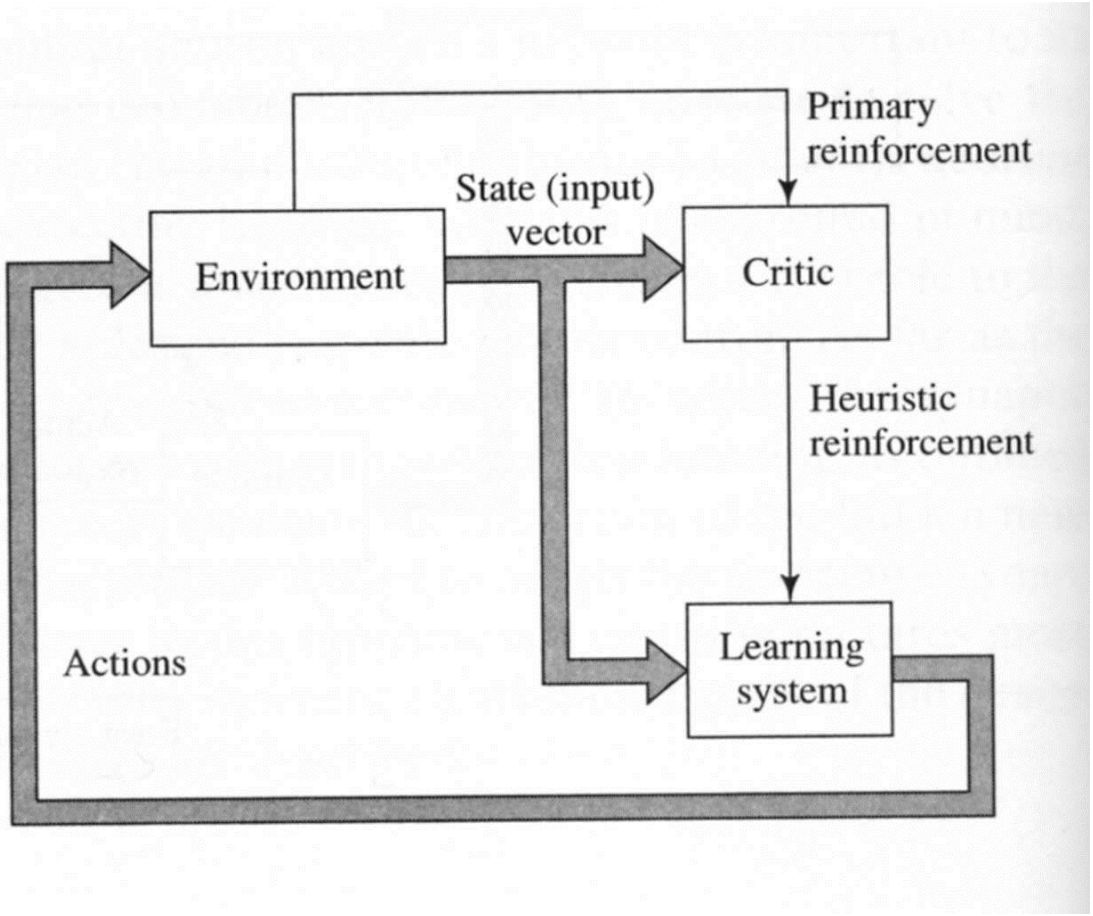
3. *The weights are adjusted by the error signals.*

What are the examples?

MLP and RBFN

NUS National University of Singapore

Learning without a teacher: Reinforcement Learning



Process of learning:

1. The neural network is interacting with the environment by taking various actions.

2. The learning system will be rewarded or penalized by its actions.

3. The weights are adjusted by the reinforcement signal.

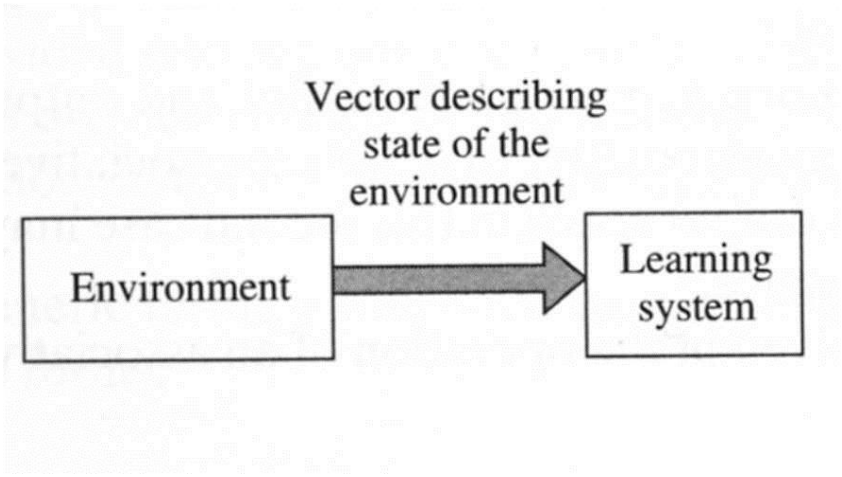
Did we ever encounter RL in our studies so far?

No.

But you are going to learn it in the second part of this course!

Learning without a teacher:
Unsupervised or self-organized learning

Process of learning:



1. The neural network is fed with input.

2. The weights are adjusted based upon the input signals only!

How can the system adjust the weights without any error or reward signal?

In MLP, we used **supervised** learning.

Is supervised learning biological plausible? Does our brain use "Backpropagation" to adjust the weights?

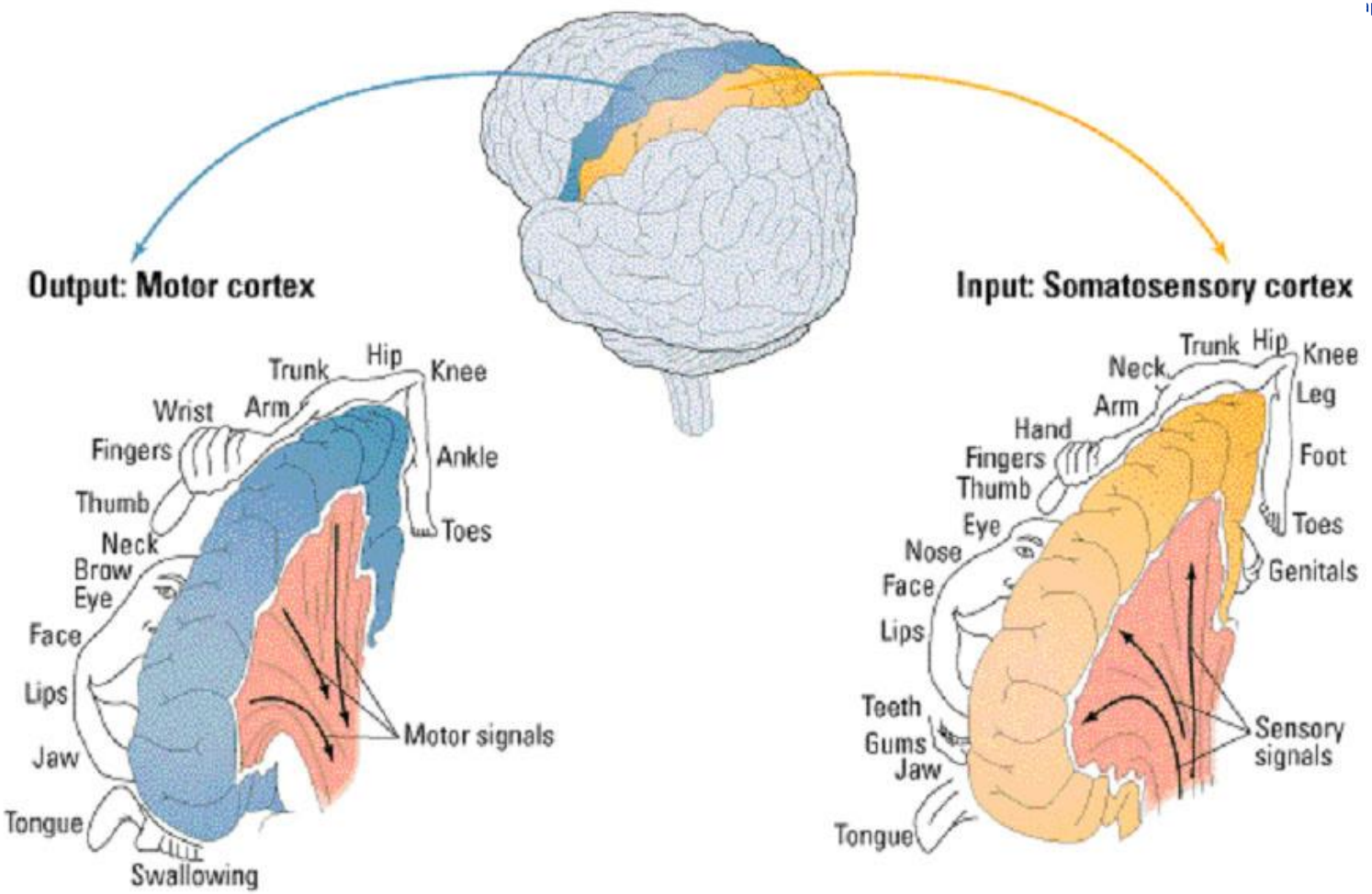
This is not biologically plausible: In a biological system, there is no external "teacher" who manipulates the network's weights from outside the network.

Biologically more adequate: **unsupervised** learning.

Reinforcement Learning is of course well biologically based.

We will study Self-Organizing Maps (SOMs) as another example for unsupervised learning, which also has a sound biological basis.

Feature Mapping of the Human Cortex

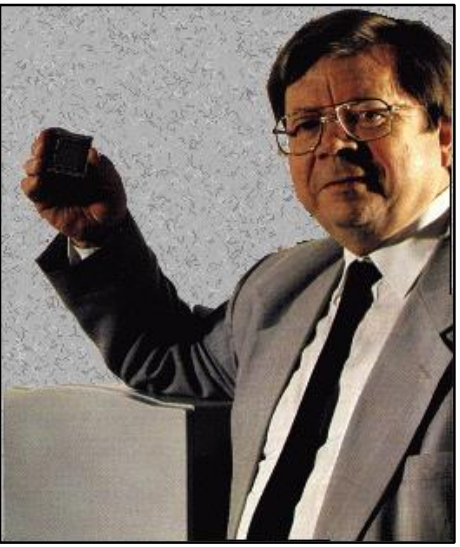


The neurons are well organized! Neighboring areas in these maps represent neighboring areas in the sensory input space or motory output space.

Is this topographical organization entirely genetically programmed?

Self-Organizing Maps: History

C. von der Malsburg
(1942—present)



Teuvo Kohonen

(1934-present)

In 1973, von der Malsburg studied the self-organizing property of the visual cortex, and concluded that:

A model of the visual cortex could not be entirely genetically predetermined; rather, a self-organizing process involving synaptic learning may be responsible for the local ordering of feature-sensitive cortical cells.

The computer simulation by von der Malsburg was perhaps the first to demonstrate self-organization. However, global topographic ordering was not achieved in his model.

The SOM is now always associated with Kohonen, who published his work on SOM in 1982, 4 years before BP.

Principle of Topographic Map Formation:

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

The neurons are spatially organized in a meaningful way!

The topology-conserving mapping can be achieved by SOMs:

- Two layers: input layer and output (map) layer

  There is no hidden layer!

- Input and output layers are fully connected.
- A topology (neighborhood relation) is defined on the output layer.

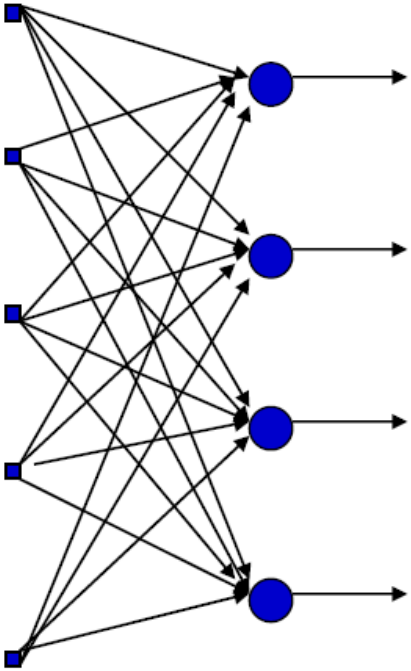
  The location of the neurons in the output layer is important!

For MLP or RBFN, do we care about where the neurons are in the output layer?

No.

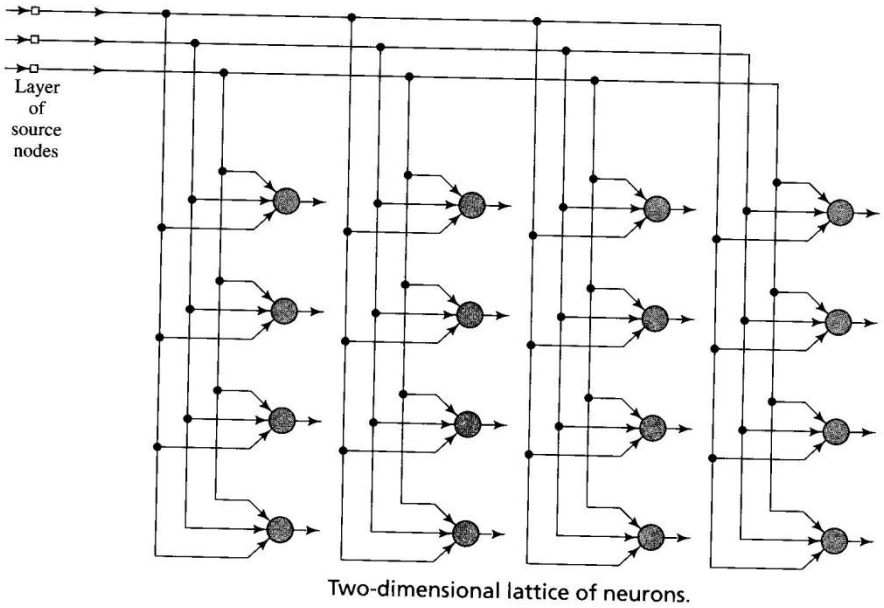
Topological organization is the unique property of SOM.

SOM----Goal



Lattice

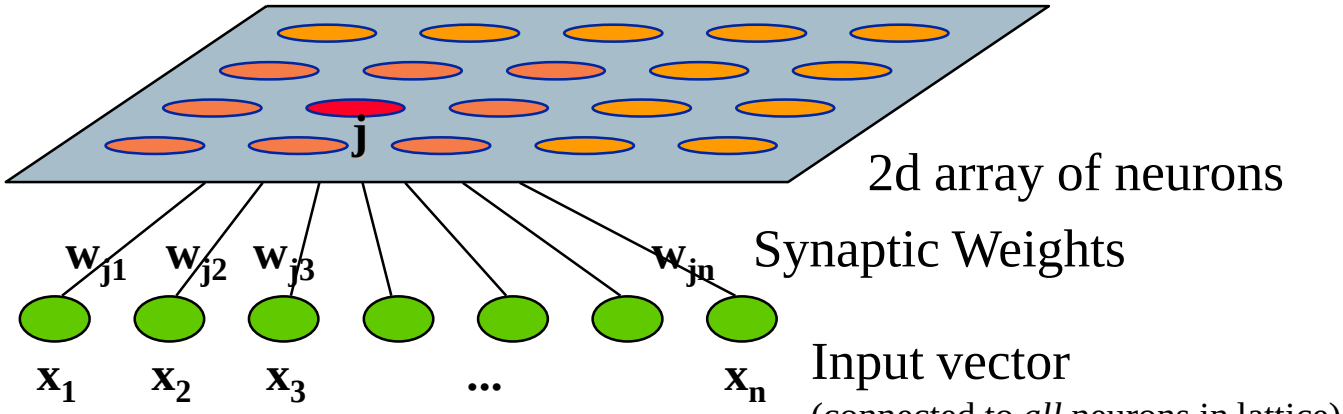
Visualization



Two-dimensional lattice of neurons.

The principle goal of the self-organizing map is to transform an incoming signal pattern of arbitary dimension into a one- or two dimensional discrete feature map, and to perform this transformation adaptively in a topologically ordered fashion.

SOM – Architecture



For each output neuron j, there are a set of synaptic weights w_{ji} connected from all the input neurons.

What is the purpose of these weights?

Do we use these weights to compute the outputs by *McCulloch and Pitts* model?

$$y_j = \varphi(\sum_{i=1}^{n} w_{ji} x_i + b_j)$$

No. In fact, there are no output signals produced by the output neurons!

The synaptic weights act as the underlying "codes" of the neurons!

Are the output neurons connected to each other by synaptic weights?

No. They are not connected by synaptic weights.

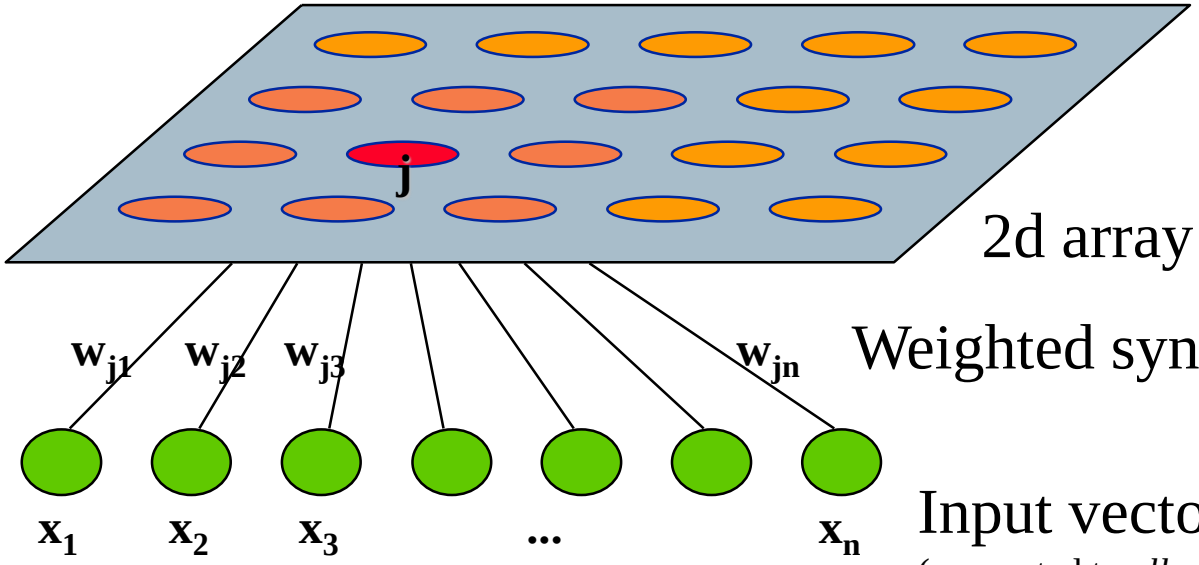
But they are related to each other by their locations in the 2D-map!

How to describe their locations in the map?

It can be simply described by their indices in the 2d-array.

A neuron with the index (i,j) in the 2d array, its location is simply (i,j) in the 2-d plane.

SOM –Algorithm Overview



2d array of neurons

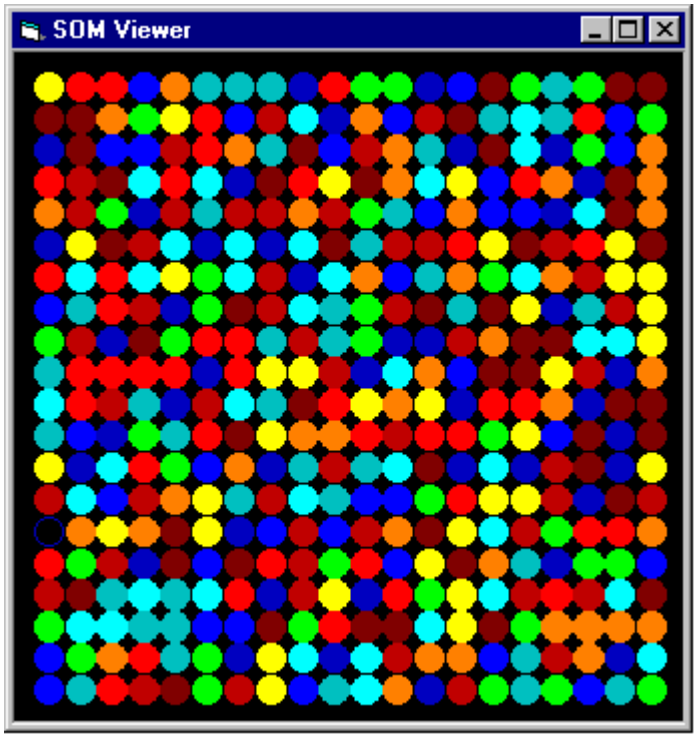
Weighted synapses

Input vector
(connected to *all* neurons in lattice)

1. **Randomly initialise all the weights.**
2. **Select input vector $x = [x_1, x_2, x_3, \ldots, x_n]$ from the training set.**
3. **Compare x with weights w_j for each neuron j to determine winner.**
4. **Update winner so that it becomes more like x, together with the winner's *neighbours*.**
5. **Adjust parameters: learning rate & 'neighbourhood function'.**
6. **Repeat from (2) until the map has converged (i.e. no noticeable changes in the weights) or pre-defined no. of training cycles have passed.**

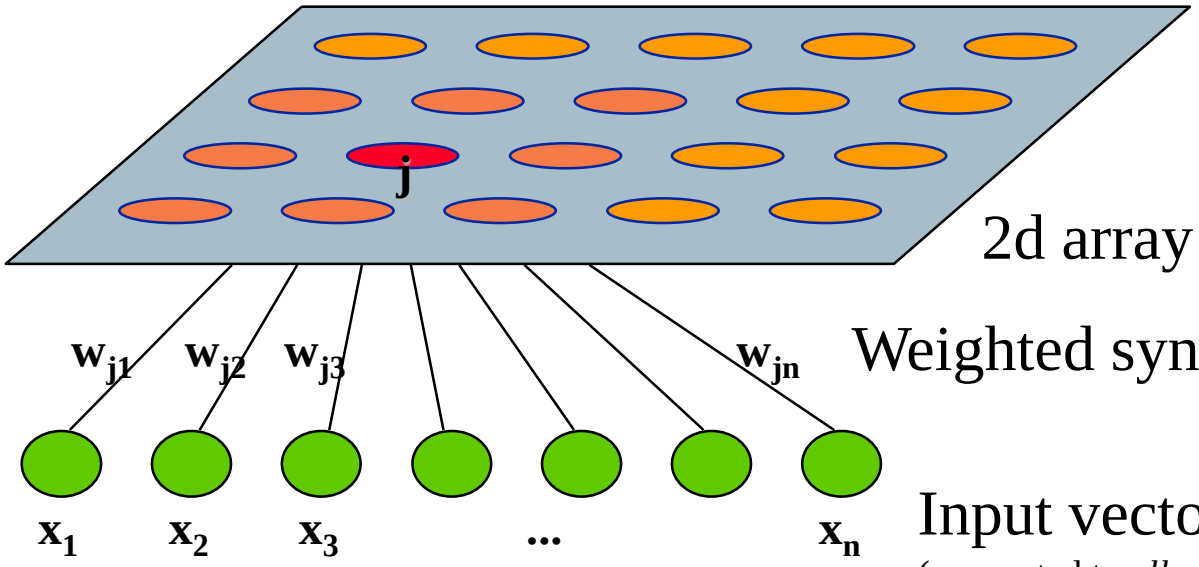
(i) Randomly initialise the weight vectors w_j for all nodes j

This can be done by assigning them small values picked from a random number generator (such as the MATLAB command “rand”).



(ii) Sampling: choose an input vector x from the training set

It can be chosen randomly from the training set (if the set is huge), or one by one in a deterministic manner like that for sequential learning for MLP.



2d array of neurons

Weighted synapses

Input vector
(connected to *all* neurons in lattice)

The formation of the self-organizing map:

1. Competition

2. Cooperation

3. Synaptic Adaptation

Competitive Process (Finding a winner)

A continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network .

Find the best-matching neuron w(x), usually the neuron whose weight vector has smallest Euclidean distance from the input vector x

$$i(x) = \arg\min_j \|x - w_j\|$$

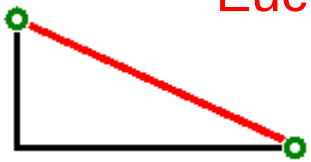
The winning neuron is that which is in some sense 'closest' to the input vector.

'Euclidean distance' is the straight line distance between the data points, if they are plotted on a (multi-dimensional) graph

Euclidean distance between two vectors a and b, $a = (a_1,a_2,\ldots,a_n)$, $b = (b_1,b_2,\ldots b_n)$, is calculated as:

$$d_{\mathbf{a,b}} = \sqrt{\sum_i (a_i - b_i)^2}$$

Euclidean distance



The winning neuron locates the center of a topological neighborhood of cooperating neurons.

How to define a topological neigborhood that is neurobiologically correct?

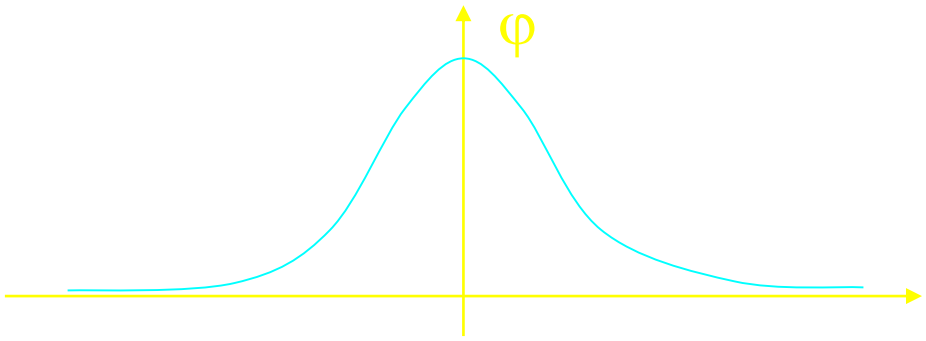
There is neurobiological evidence for *lateral interaction:*

A neuron that is firing tends to excite the neurons in its immediate neighborhood more than those farther away from it.

The closer to the winner neuron, the more impact it receives.

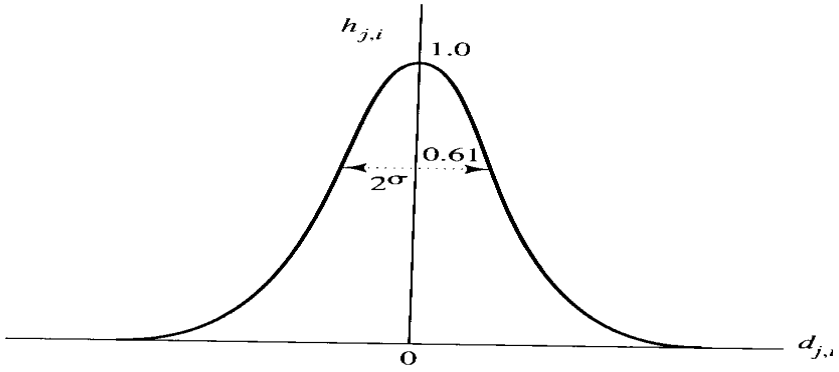
From the biological evidence,

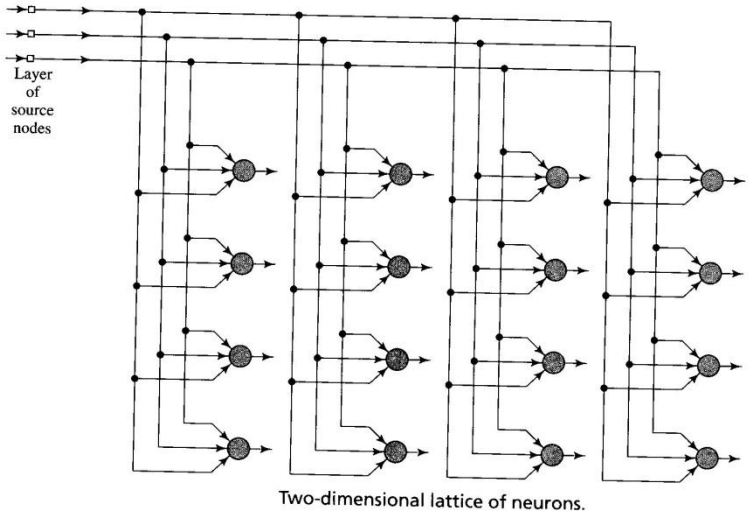
1. **The topological neighborhood is symmetric about the maximum point (the winner neuron).**
2. **The amplitude of the topological neighborhood decreases monotonically with the increasing lateral distance.**



What is the natural choice of the function to describe this bell shape?

The typical choice of topological neighborhood function





$$h_{j,i(x)} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right)$$

$d_{j,i}$ the Euclidian distance from the neuron j to the winning neuron i (associated with the input vector x).

The position of the neuron is described by the index of the neuron in the matrix (2d-lattice). For instance, the distance from the neuron at position (l, m) to the one at (n,k) is

$$d_{j,i} = \sqrt{(l-n)^2 + (m-k)^2}$$

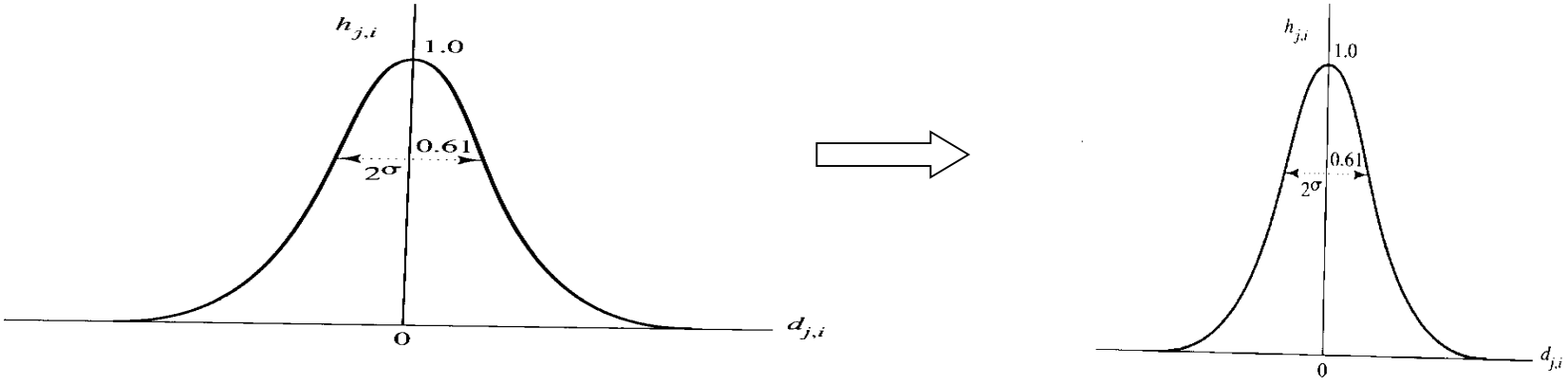
$h_{j,i(x)}$ is a measure of the effectiveness of the winner on its neighbors.

σ "effective width" of the topological neighborhood.

When the neuron is σ away from the winner, $h_{j,i}=\exp(-0.5)=0.61$, it is heavily affected.

When the neuron is 2σ away from the winner, $h_{j,i}=\exp(-2)=0.14$, it is less affected.

Another unique feature of the SOM algorithm is that the size of the topological neighborhood shrinks with time.



What is the difference between the two Gaussians above? The effective width!

"effective width" decreases with time. A popular choice is:

Time-varying width

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1}) \qquad n = 0,1,2,...,$$

τ_1 is a time-constant to control the decay rate of the effective width.

Time-varying neighborhood function:

$$h_{j,i(x)}(n) = \exp(-\frac{d_{j,i}^2}{2\sigma(n)^2})$$

It means that the influence of the winner on its neighbors decreases with time.

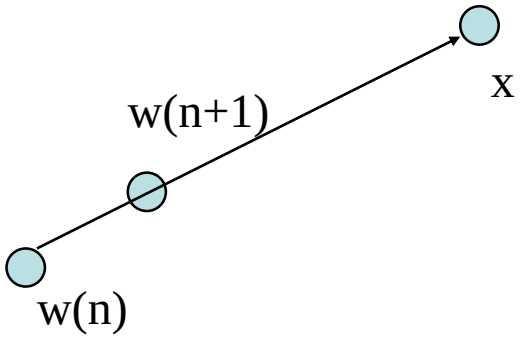
Adaptation

$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n)(x - w_j(n)), \qquad j = 1,\ldots,M$$

What is the geometrical meaning of this algorithm?

$$w_j(n+1) = (1-\alpha) w_j(n) + \alpha x, \qquad \alpha = \eta(n) h_{j,i(x)}(n) > 0$$

The new weight is the weighted average of the input and the current weight!



The synaptic weight vector w_i of winning neuron i moves toward the input vector x!

All the neurons in the neighborhood of the winning neuron also move toward the input vector! The farther away, the less change.

Upon repeated presentations of the training data, the synaptic weight vector tend to **follow the distribution of the input vectors** due to the neighborhood updating.

The algorithm leads to a topological ordering of the feature map in the sense that neurons that are adjacent in the lattice will tend to have similar synaptic weight vectors.

NUS National University of Singapore

Adaptation---Learning Rate

$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n)(x - w_j(n)), \qquad j = 1,\ldots,M$$

Would it converge if the learning rate is a constant?

No. The winner neuron will always update its weights since although its effect on its neighbors decreases with time. $h_{\mathrm{i(x)},i(x)}=1$

How to make sure that the weights converge?

The learning-rate parameter η(n) should also decrease with time.

One possible choice: $\eta(n) = \eta_0 \exp(-\frac{n}{\tau_2}), \qquad n = 0,1,2,\ldots,$

τ_2 is another time-constant to control the decay rate of the learning rate.

The first phases of the adaptive process

The adaptation of the weights is decomposed into two phases: an ordering or self-organizing phase followed by a convergence phase.

1. Self-organizing or ordering phase

The ordering phase may take as many as 1000 iterations, and possibly more.

•The learning rate η(n) should begin with a value close to 0.1; there after it should decrease gradually, but remain above 0.01.

$$\eta_0 = 0.1 \quad \tau_2 = T \qquad \rightarrow \eta(n) = 0.1\exp(-\frac{n}{T}), \qquad n = 0,1,2,...,T$$

T is the total number of iterations for the first phase.

The neighborhood function should initially include almost all neurons, and then shrink with time. We may set the initial size of the width equal to the "radius" of the lattice. For instance if the size of the lattice is MxN, then the initial width can be set as

$$\sigma_0 = \frac{\sqrt{M^2 + N^2}}{2}$$

Correspondingly, the time constant can be chosen as $\tau_1 = \frac{T}{\log(\sigma_0)}$

$$\sigma(n) = \sigma_0 \exp(-\frac{n}{\tau_1}) \qquad n = 0,1,2,...,T$$

At the end, $n = T \Longrightarrow \sigma(T) = 1$

The winner only affects its immediate neighbors at the end of first phase!

The second phase of the adaptive process

2. Convergence phase: This second phase is needed to fine tune the feature map.

As a general rule, the number of itertations must be at least 500 times the number of neurons in the network. Thus, the convergence phase may have to go on for thousands and possibly tens of thousands of iterations.

For good statistical accuracy, the learning parameter $\eta(n)$ should be maintained at a small value, on the order of 0.01. In any event, it must not be allowed to decrease to zero.

The neighborhood function should contain only the nearest neighbors of a winning neuron, which may eventually reduce to one or zero neighboring neurons.

In many applications, the second phase is not needed if convergence of the parameters are not critical.

Summary of the Algorithm

For n-dimensional input space and m output neurons:

(1) Randomly initialize the weight vector w_i for neuron i, i = 1, ..., m

(2) Sampling: choose an input vector x from the training set.

(3) Determine winner neuron k:

$\|w_k - x\| = \min_i \|w_i - x\|$ (Euclidean distance)

(4) Update all weight vectors of all neurons i in the neighborhood of the winning neuron i(x):

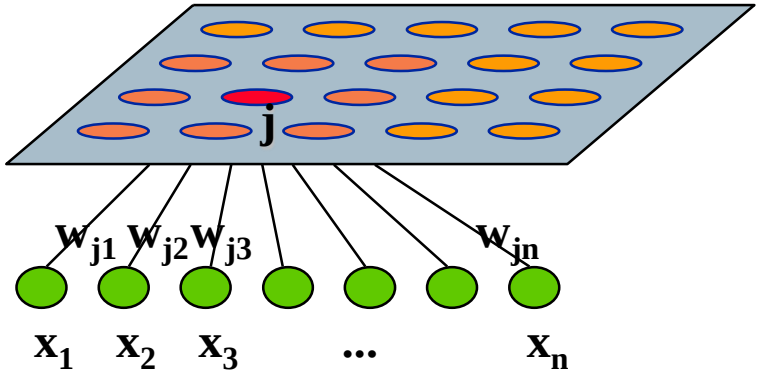
$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n)(x - w_j(n)), \qquad j = 1, \ldots, m$$

(5) If convergence criterion met, STOP.
Otherwise, go to (2).

Isn't it a simple algorithm? It is the simplest compared to MLP and RBFN.

The Kohonen's SOM algorithm is so simple to implement, yet mathematically so difficult to analyze its properties in a general setting. It is still a hot research topic.

How to use SOM?



After training the network, how to use the map?

Question: how do you find out whether the neurons are organized in a meaningful way?

What does each neuron stand for?

There are many possible ways to interpret the neurons. The simplest one is just finding out which input signal (in the training set) stimulates the particular neuron the most.

Determine winner input signal for neuron j:

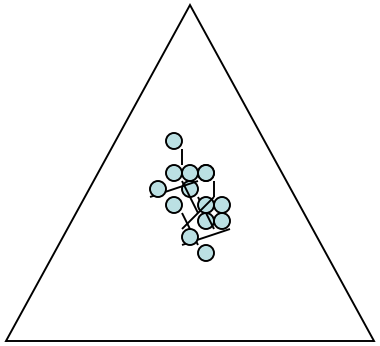
$\|x_k - w_j\| = \min_i \|x_i - w_j\|$ (Euclidean distance)

Mark each neuron by the particular input signal for which it produces the best response.

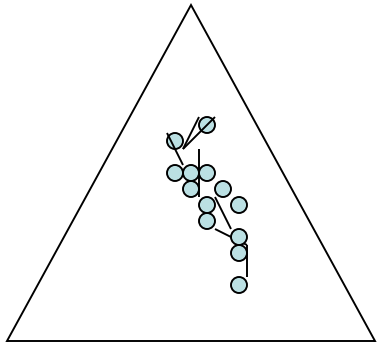
The resulting map is so called the “contextual map” or “semantic map”.

Example I: Learning a one-dimensional representation of a two-dimensional (triangular) input space:

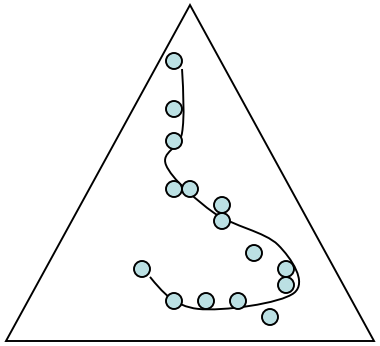
In this case, the topological map is one-dimensional.



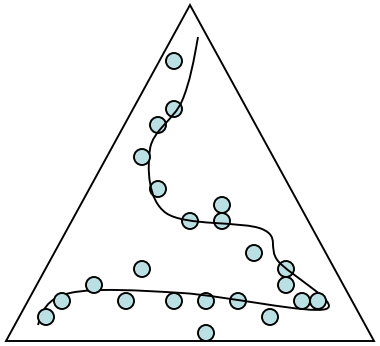
0



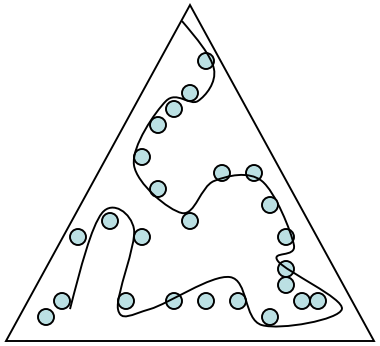
20



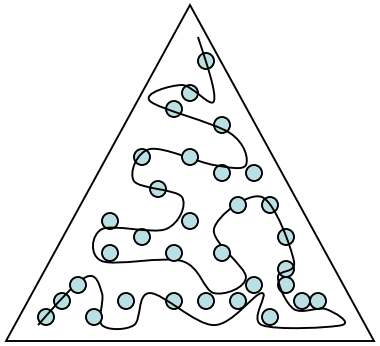
100



1000



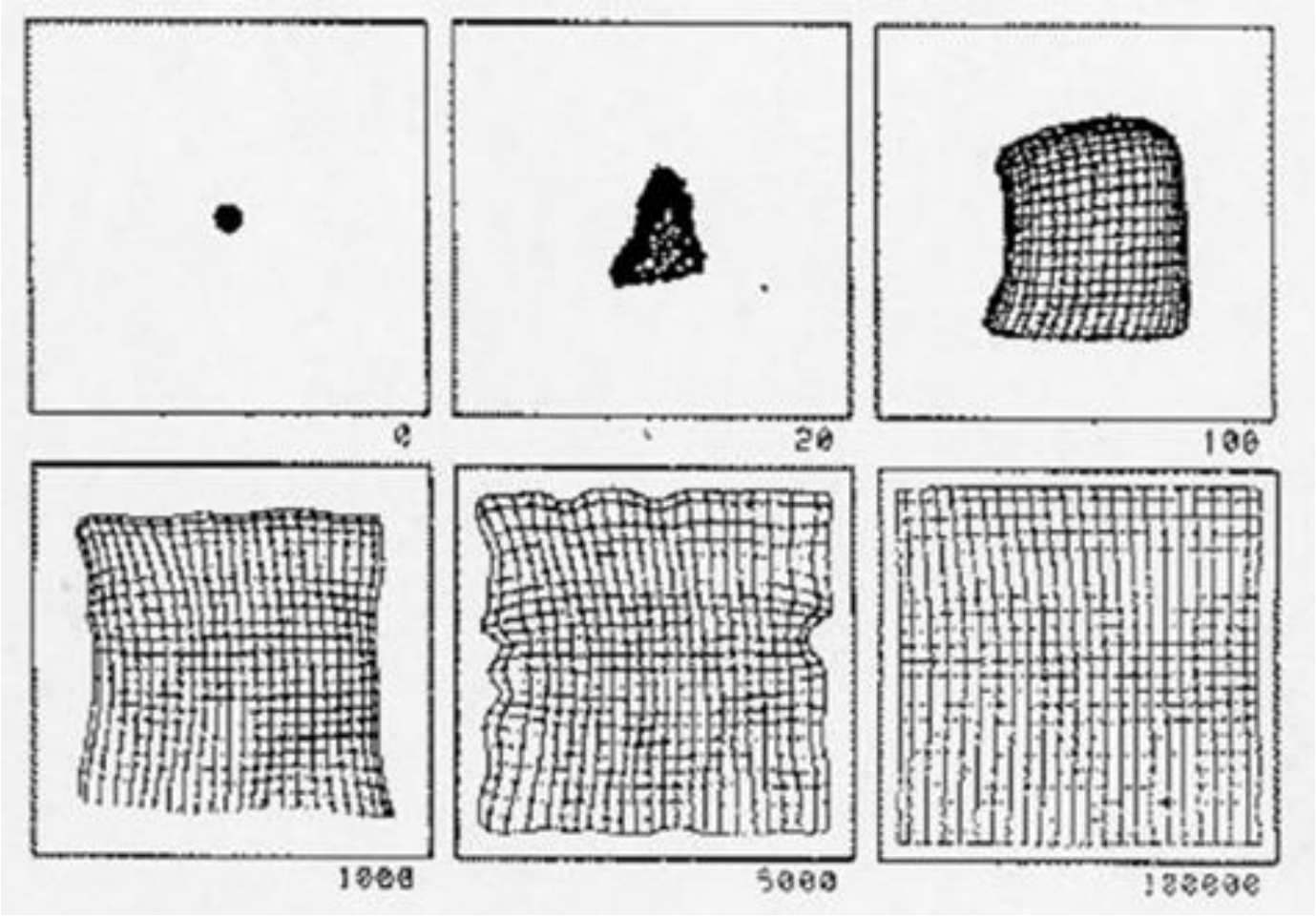
10000



25000

Example II: Learning a two-dimensional representation of a two-dimensional (square) input space:

In this case, the neurons are organized in a 2-d lattice (most common way).



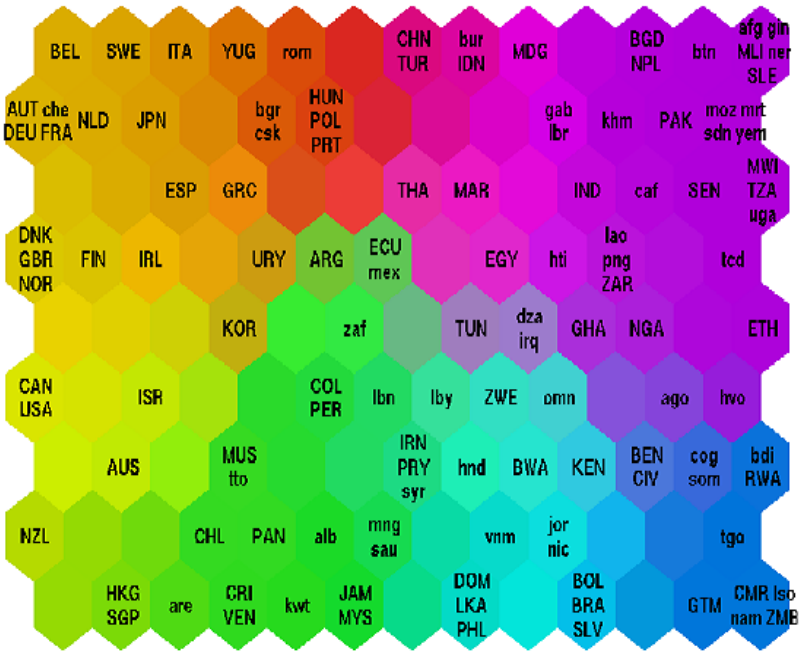
demo

How about high-dimensional input?

Example III: Learning a two-dimensional mapping of texture images

The inputs are texture images. The SOM is a 10x8 map.

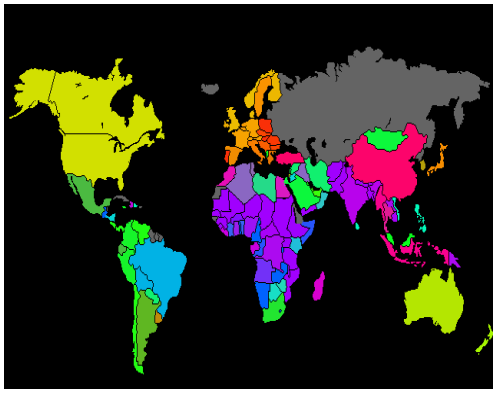
Example IV: Classifying World Poverty



The Country Names

AFG	Afghanistan	GTM	Guatemala	NZL	New Zealand
AGO	Angola	HKG	Hong Kong	OAN	Taiwan, China
ALB	Albania	HND	Honduras	OMN	Oman
ARE	United Arab Emirates	HTI	Haiti	PAK	Pakistan
ARG	Argentina	HUN	Hungary	PAN	Panama
AUS	Australia	IDN	Burkina Faso	PER	Peru
AUT	Austria	IDN	Indonesia	PHL	Philippines
BDI	Burundi	IND	India	PNG	Papua New Guinea
BEL	Belgium	IRL	Ireland	POL	Poland
BEN	Benin	IRN	Iran, Islamic Rep.	PRT	Portugal
BGD	Bangladesh	IRQ	Iraq	PRY	Paraguay
BGR	Bulgaria	ISR	Israel	ROM	Romania
BOL	Bolivia	ITA	Italy	RWA	Rwanda
BRA	Brazil	JAM	Jamaica	SAU	Saudi Arabia
BTN	Bhutan	JOR	Jordan	SDN	Sudan
BUR	Myanmar	JPN	Japan	SEN	Senegal
BWA	Botswana	KEN	Kenya	SGP	Singapore
CAF	Central African Rep.	KHM	Cambodia	SLE	Sierra Leone
CAN	Canada	KOR	Korea, Rep.	SLV	El Salvador
CHE	Switzerland	KWT	Kuwait	SOM	Somalia
CHL	Chile	LAO	Lao PDR	SWE	Sweden
CHN	China	LBN	Lebanon	SYR	Syrian Arab Rep.
CIV	Cote d'Ivoire	LBR	Liberia	TCD	Chad
CMR	Cameroon	LBY	Libya	TGO	Togo
COG	Congo	LKA	Sri Lanka	THA	Thailand
COL	Colombia	LSO	Lesotho	TTO	Trinidad and Tobago
CRI	Costa Rica	MAR	Morocco	TUN	Tunisia
CSK	Czechoslovakia	MDG	Madagascar	TUR	Turkey
DEU	Germany	MEX	Mexico	TZA	Tanzania
DNK	Denmark	MLI	Mali	UGA	Uganda
DOM	Dominican Rep.	MNG	Mongolia	URY	Uruguay
DZA	Algeria	MOZ	Mozambique	USA	United States
ECU	Ecuador	MRT	Mauritania	VEN	Venezuela
EGY	Egypt, Arab Rep.	MUS	Mauritius	VNM	Viet Nam
ESP	Spain	MWI	Malawi	YEM	Yemen, Rep.
ETH	Ethiopia	MYS	Malaysia	YUG	Yugoslavia
FIN	Finland	NAM	Namibia	ZAF	South Africa
FRA	France	NER	Niger	ZAR	Zaire
GAB	Gabon	NGA	Nigeria	ZMB	Zambia
GBR	United Kingdom	NIC	Nicaragua	ZWE	Zimbabwe
GHA	Ghana	NLD	Netherlands		
GIN	Guinea	NOR	Norway		
GRC	Greece	NPL	Nepal		

Helsinki University of Technology

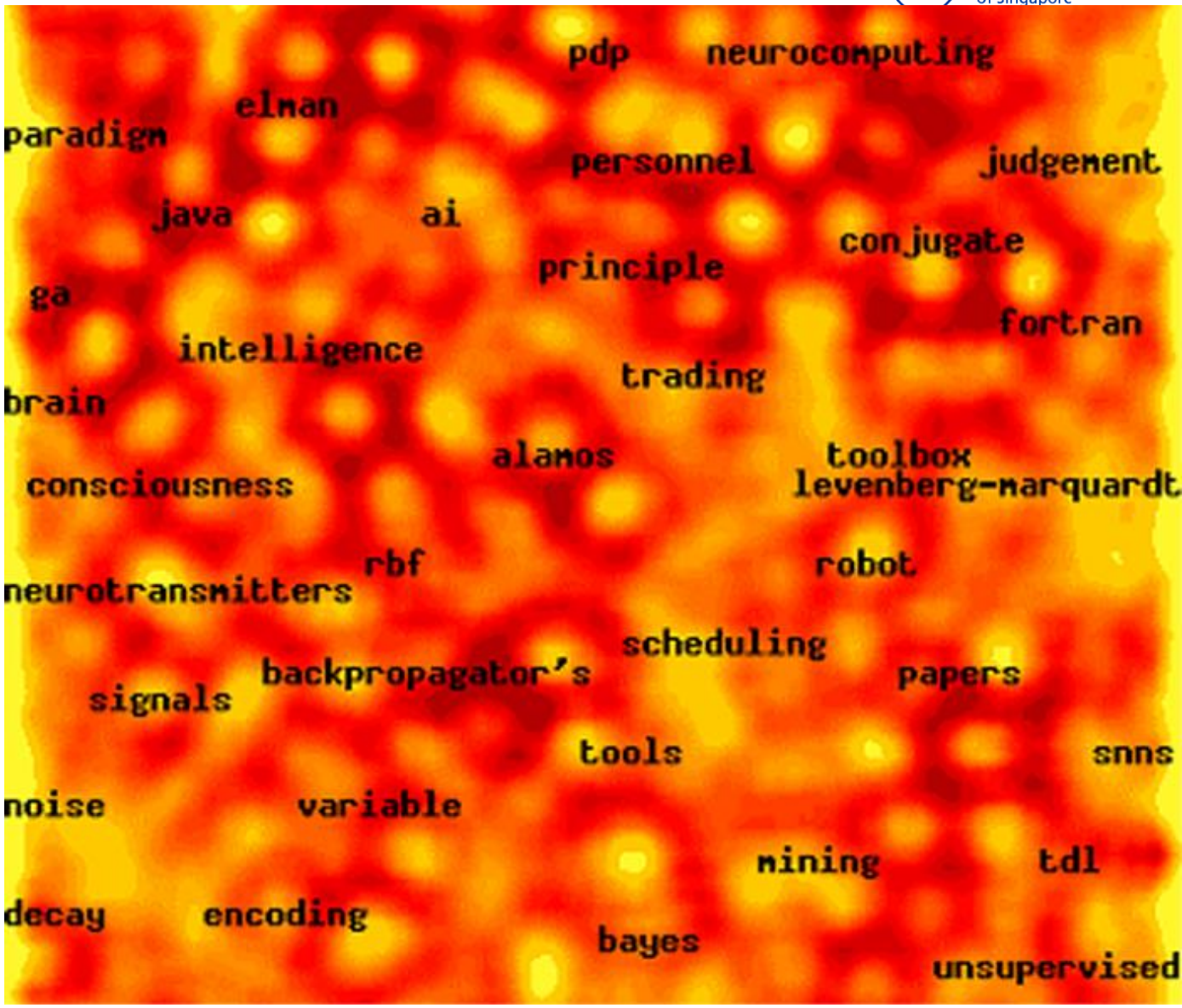


'Poverty map' based on 39 indicators from World Bank statistics (1992)

Example V: WEB SOM

- SOM analysis technique to map thousands of articles posted on Usenet newsgroups

Lagus et al. (1996); Honkela et al. (1998) - HUT NN Research Centre)

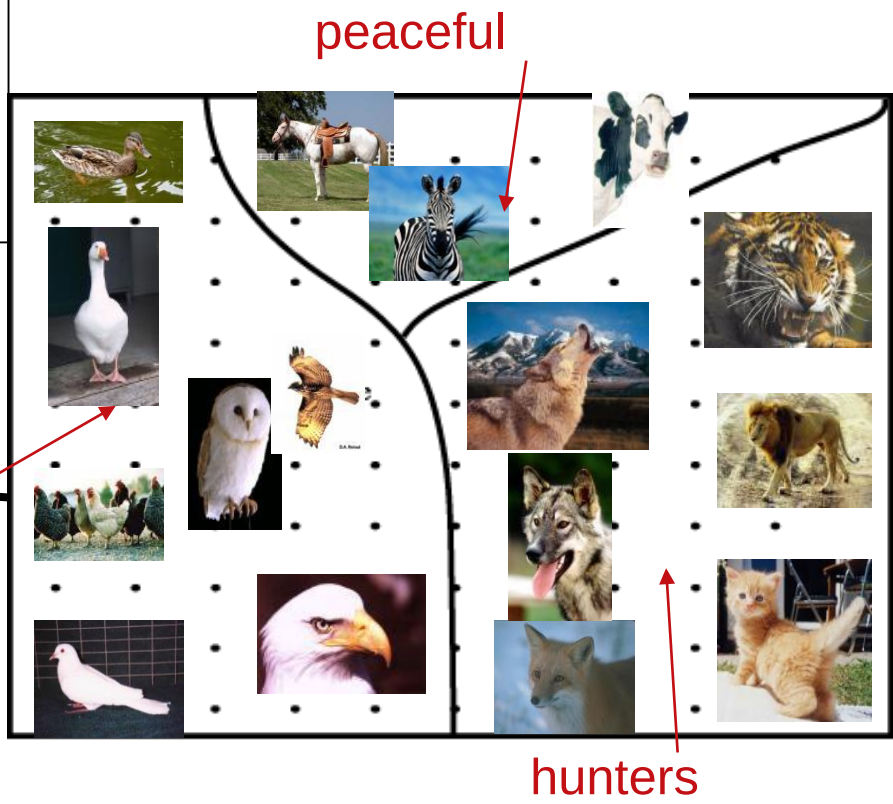


Example VI: Contextual Maps by SOM

Animal names and their attributes

		Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
is	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
	Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	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
	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
likes to	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

A grouping according to similarity has emerged



SOM can really organize the data!

[Teuvo Kohonen 2001] Self-Organizing Maps; Springer;

NUS National University of Singapore

Break

- Real life terminator robot (after 1hr)

What is Neural Network (NN)?

A neural network is a *massively parallel distributed processor* made up of simple processing unit, which has a natural propensity for *storing experiential knowledge* and making it available for use.

It employs a massive inter-connection of "simple" computing units - *neurons*.

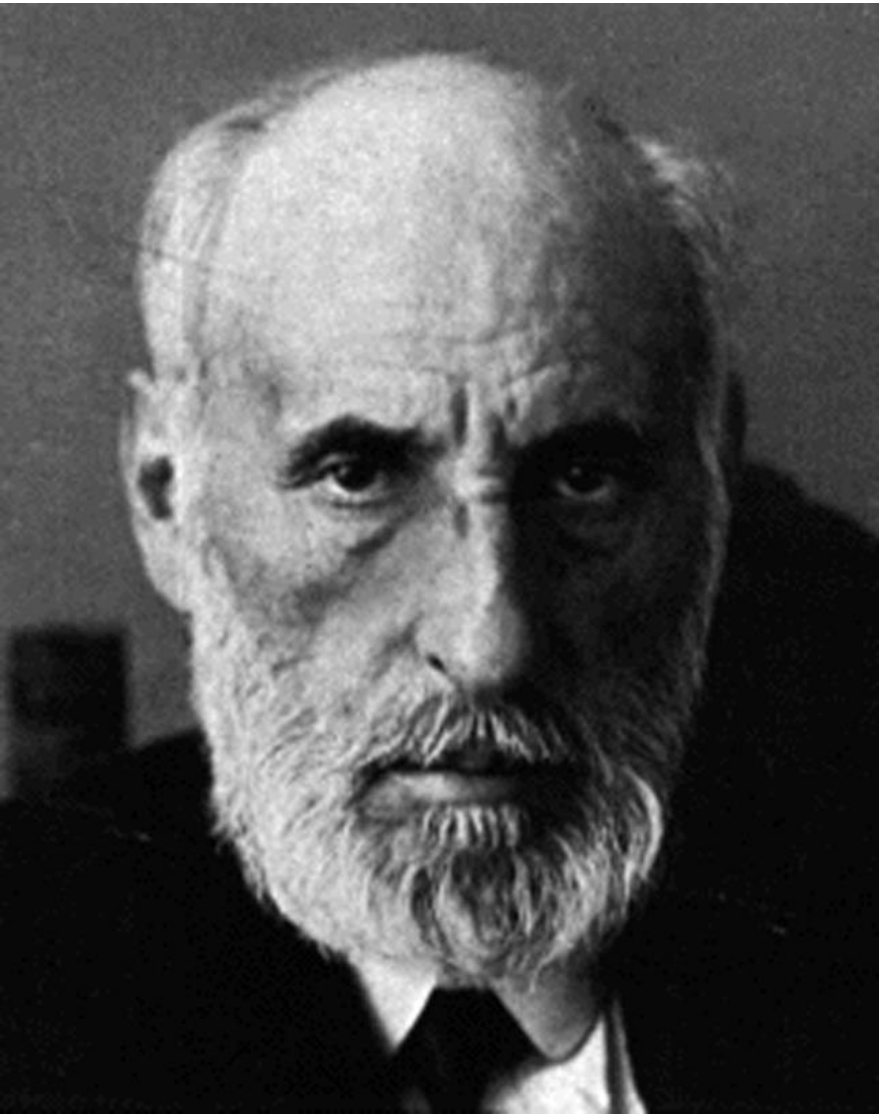
It is capable of organizing its structure consists of many neurons, to perform tasks that are many times faster than the fastest digital computers nowadays.

Knowledge is obtained from the data/input signals provided.

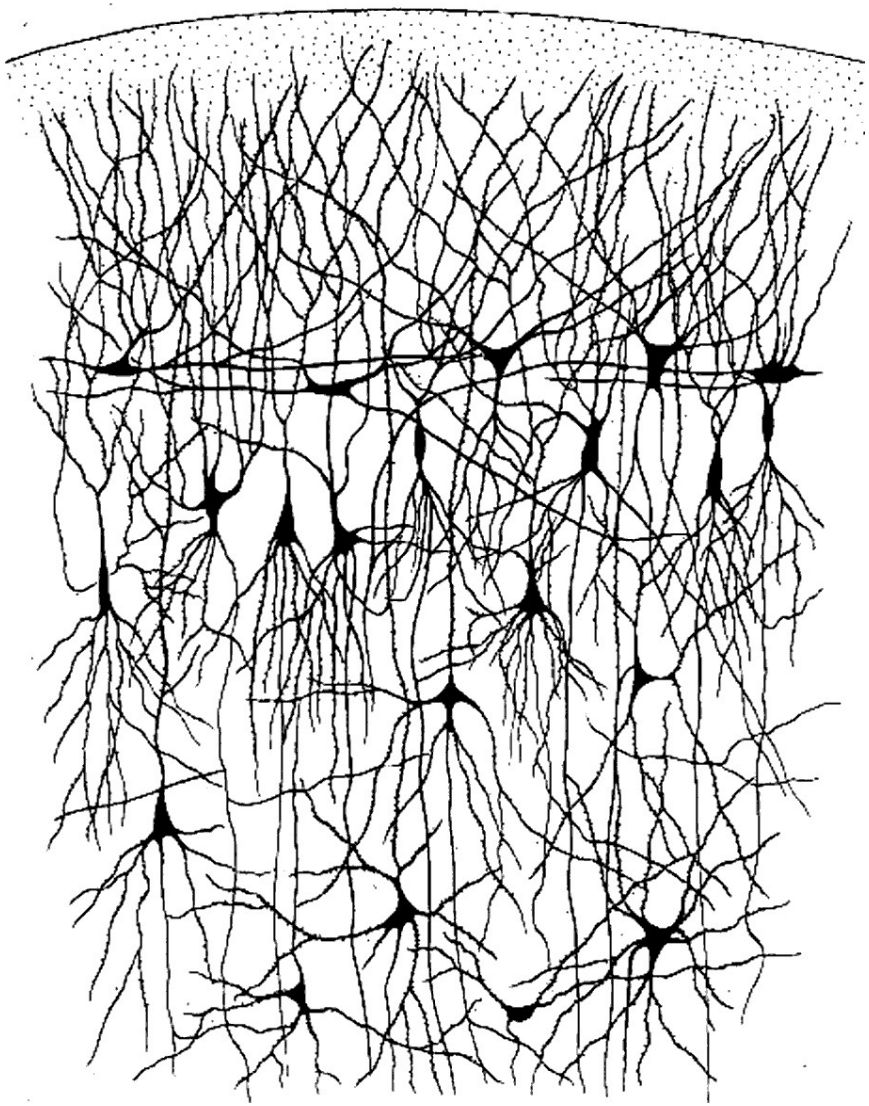
Knowledge is learned by adjusting the synapses!

The artificial neural networks are largely inspired by the biological neural networks, and the ultimate goal of building an intelligent machine which can mimic the human brain.

The understanding of neuron started more than 100 years ago:



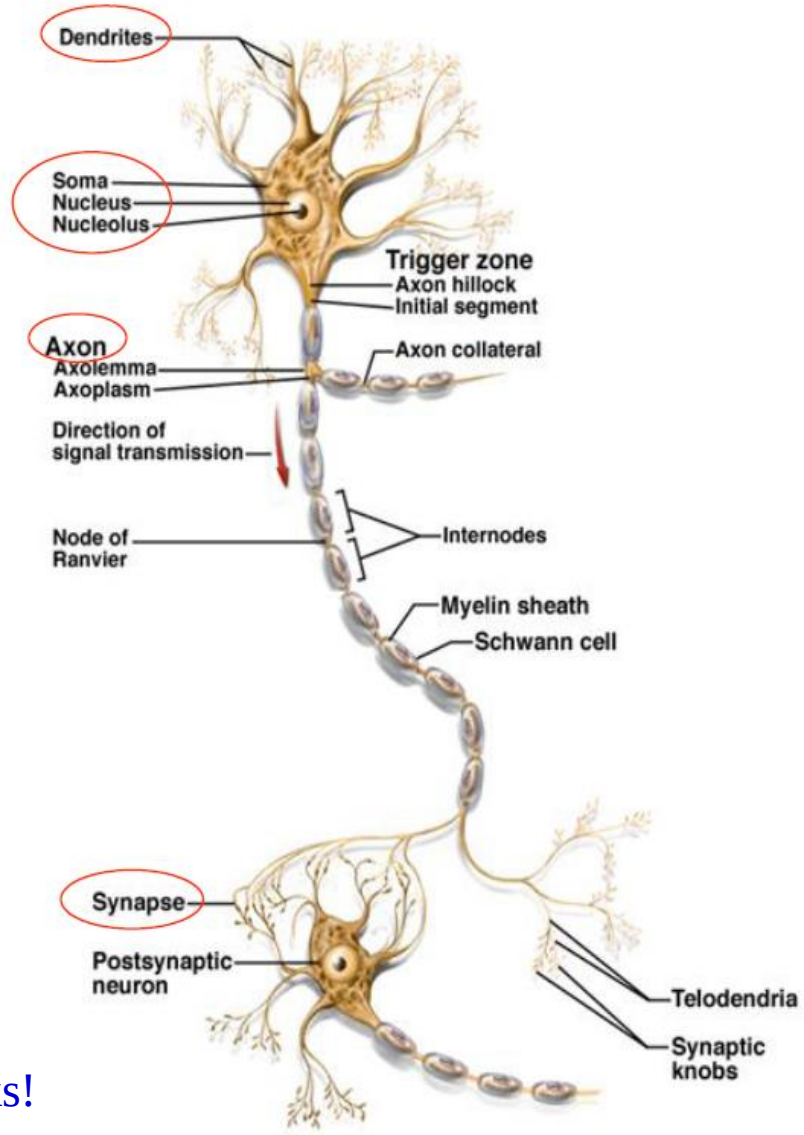
Santiago Ramon y Cajal 1852-1934



Biological Neuron

The major job of neuron:

1. It receives information, usually in the form of electrical pulses, from many other neurons.
2. It does what is, in effect, a complex dynamic sum of these inputs.
3. It sends out information in the form of a stream of electrical impulses down its axon and on to many other neurons.
4. The connections (synapses) are crucial for excitation, inhibition or modulation of the cells.
5. Learning is possible by adjusting the synapses!

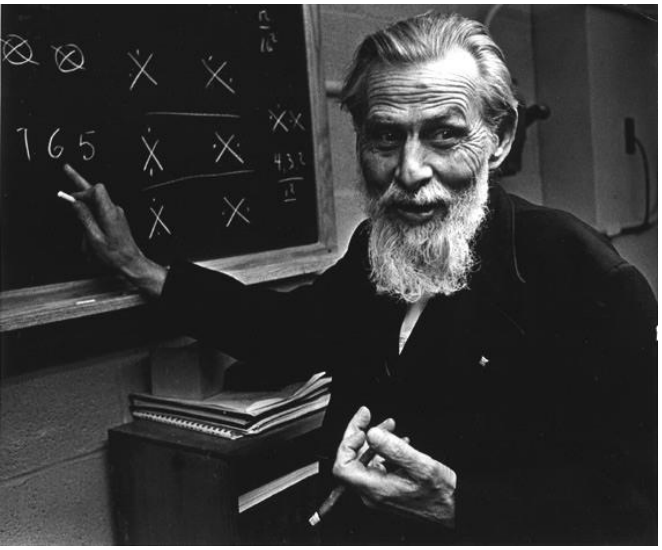


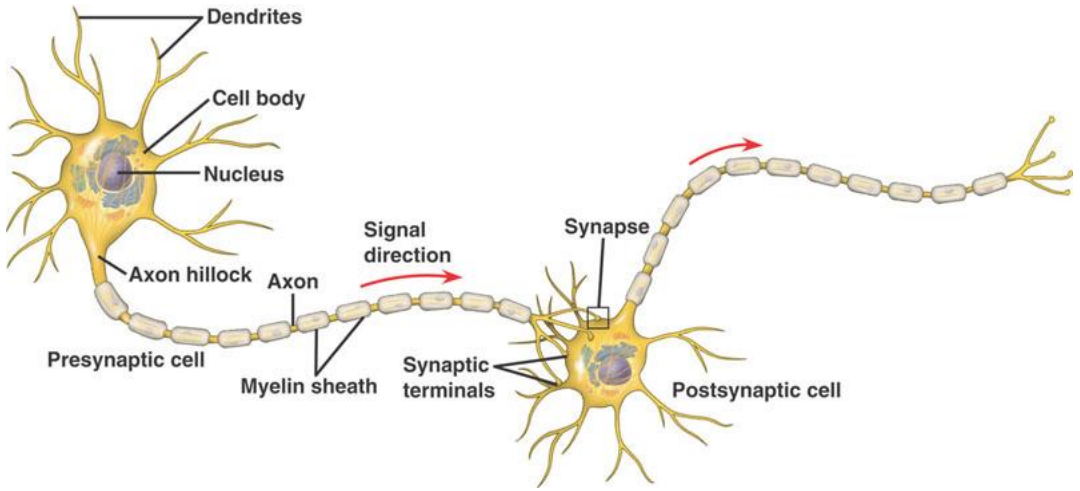
How to build a mathematical model of the neuron?

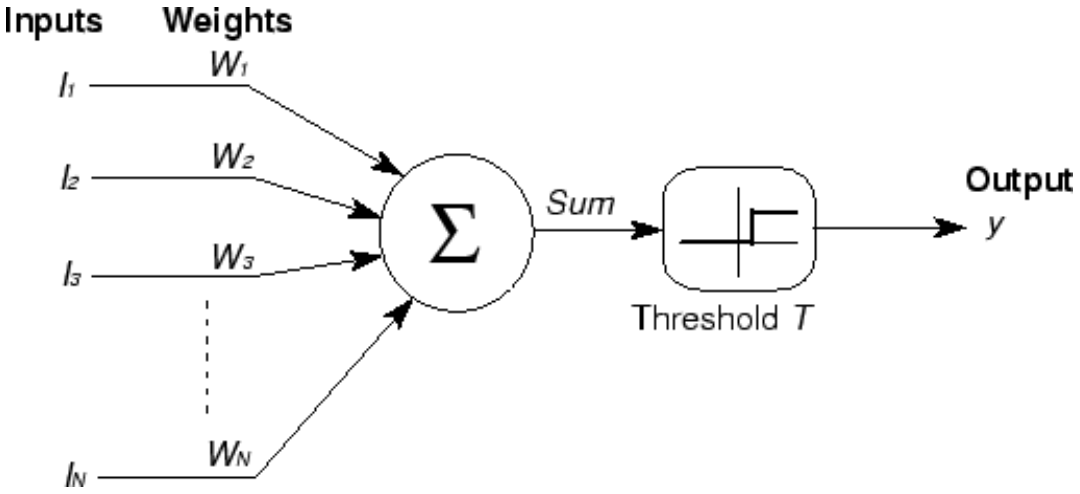
This is the starting point of the artificial neural networks!

The beginning of the artificial neural networks

McCulloch and Pitts, 1943



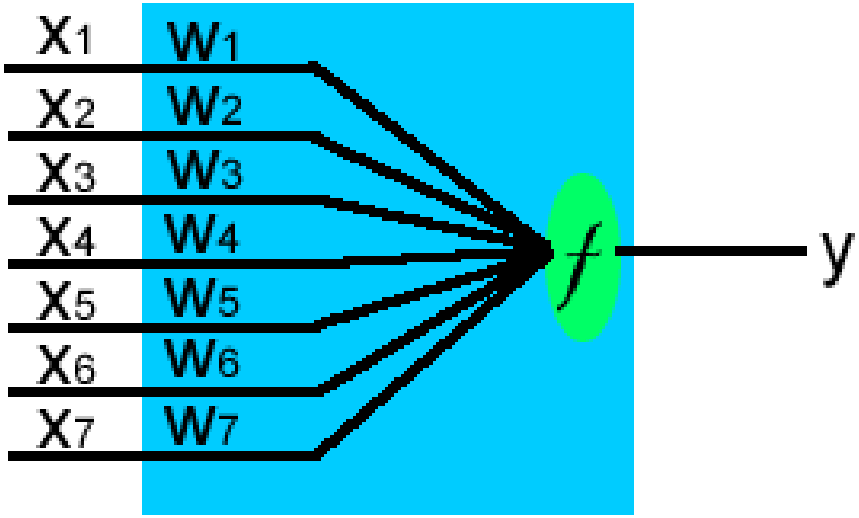




The next major step: Perceptron—single layer neural networks

Frank Rosenblatt, 1958

Frank Rosenblatt
(1928-1969)



Supervised learning:

$$w(n+1) = w(n) + \eta e(n) x(n)$$

$$e(n) = d(n) - y(n)$$

The weights were initially random. Then it could learn to perform certain simple tasks in pattern recognition.

Rosenblatt proved that for a certain class of problems, it could learn to behave correctly! During 1960s', it seemed that neural networks could do anything.

Perceptron Learning Algorithm

Start with a randomly chosen weight vector ***w(1)***;

Update the weight vector to

$$w(n+1) = w(n) + \eta e(n) x(n)$$
$$e(n) = d(n) - y(n)$$

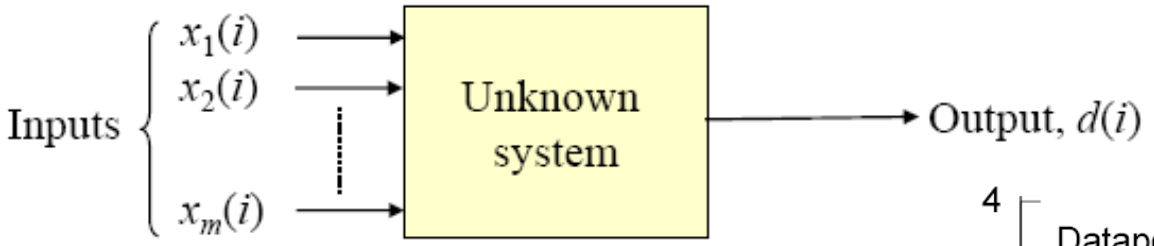
where $\eta > 0$ and $d = \begin{cases} 1 \text{ if } \boldsymbol{x} \text{ belongs to class } C_1 \\ 0 \text{ if } \boldsymbol{x} \text{ belongs to class } C_2 \end{cases}$

Perceptron Convergence Theorem: (Rosenblatt, 1962)

If C_1 and C_2 are linearly separable, then the perceptron training algorithm "converges" in the sense that after a *finite number* of steps, ***the synaptic weights*** remain unchanged and the perceptron correctly classifies all elements of the training set.

Regression Problem

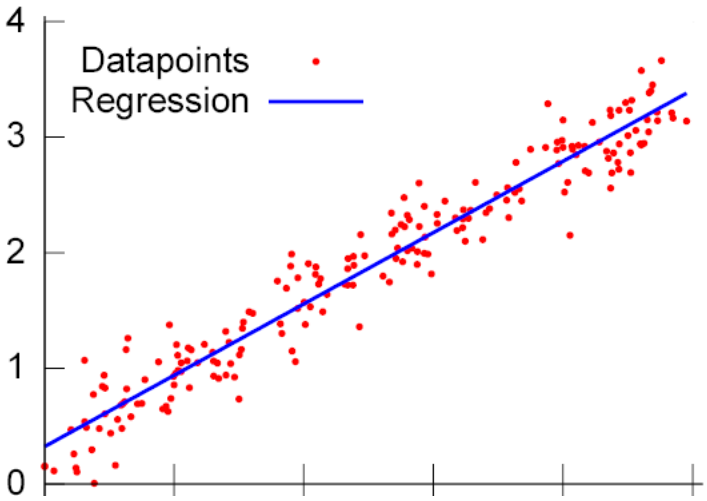
Consider a multiple input-single output system whose mathematical characterization is unknown:



Given a set of observations of input-output data:

$$T: \{\mathbf{x}(i), d(i); i = 1, 2, \ldots, n\}$$

where $\mathbf{x}(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$



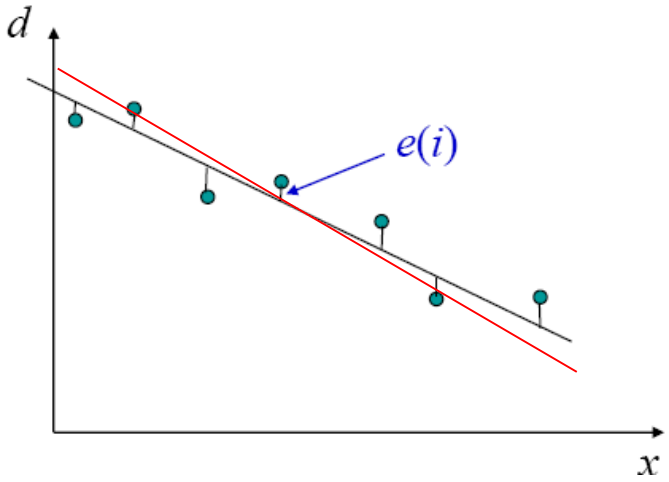
m = dimensionality of the input space; i = time index.

How to design a model of the unknown system?

Optimization problem: Minimize the cost function!

What is the most common cost function?

$$E(w)=\sum_{i=1}^{n}e(i)^2=\sum_{i=1}^{n}(d(i)-y(i))^2$$



What is the Optimality Condition? $\nabla\left(E(\boldsymbol{w}^*)\right)=0$

There are two ways to solve the problem:

If the model is simple, then directly solve $\nabla\left(E(\boldsymbol{w}^*)\right)=0$

Iterative descent algorithm**:** Starting with an initial guess denoted by $\boldsymbol{w}(0)$, generate a sequence of weight vectors $\boldsymbol{w}(1)$, $\boldsymbol{w}(2)$,…, such that the cost function $E(\boldsymbol{w})$ is reduced at each iteration of the algorithm, as shown by

$$E(\boldsymbol{w}(n+1)) < E(\boldsymbol{w}(n))$$

How to choose the iterative algorithm

$$w(n+1)=w(n)+\Delta w(n)$$

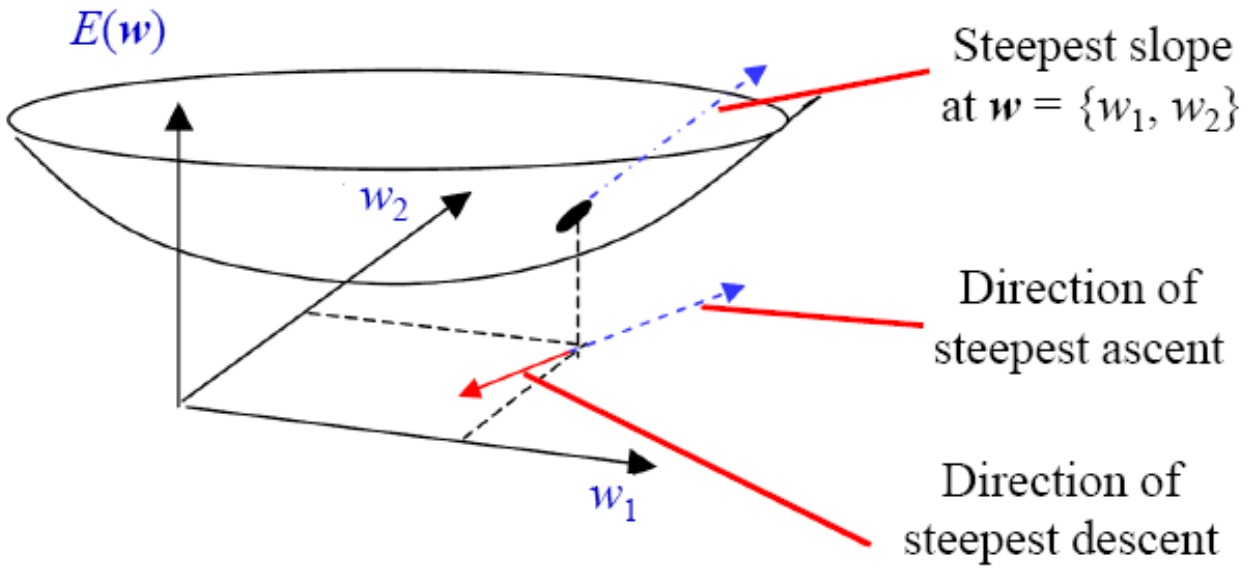
such that the cost is always decreasing ?

What is the simplest way if the gradient is known?

Method of Steepest Descent (Gradient Descent)

$$w(n+1) = w(n) + \Delta w(n)$$

Successive adjustment applied to the weight vector $\boldsymbol{w}$ are in the direction of steepest descent (a direction opposite to the gradient vector $\nabla E(\boldsymbol{w})$).



Let $\boldsymbol{g(n)} = \nabla E(\boldsymbol{w(n)})$, steepest descent algorithm is formally described by

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \cdot \boldsymbol{g}(n)$$

where η is a positive constant called the stepsize or learning-rate

Is there any condition on the learning-rate to make the algorithm work?

The rate has to be sufficiently small!

Linear Regression Problem

Consider that we are trying to fit a linear model to a set of input-output pairs ($\boldsymbol{x}(1)$, $d(1)$), ($\boldsymbol{x}(2)$, $d(2)$) …, ($\boldsymbol{x}(n)$, $d(n)$) observed in an interval of duration n.

$$y(x)=w_1x_1+w_2x_2+\ldots+w_nx_n+b$$

Cost Function: $E(w)=\sum_{i=1}^{n}e(i)^2=\sum_{i=1}^{n}(d(i)-y(i))^2$

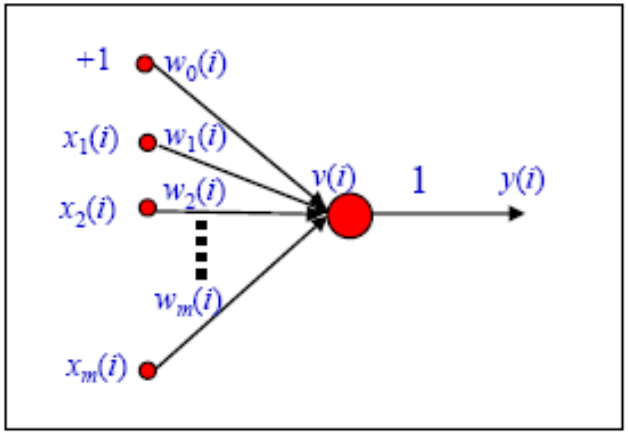
Of course, the answer can be easily found by solving $\nabla\left(E(\boldsymbol{w}^*)\right)=0$ directly.

Standard Linear Least Squares

$$w=(X^TX)^{-1}X^Td$$

The Least-Mean-Square algorithm:

$$e(n)=d(n)-w^T(n)x(n)$$
$$w(n+1)=w(n)+\eta e(n)x(n)$$



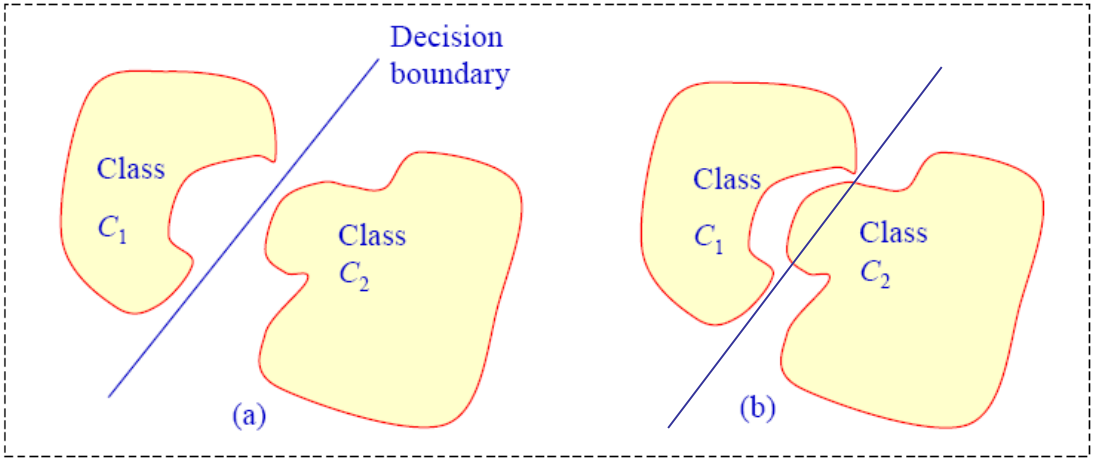
Does it take the same form as that for the Perceptron?

Yes.

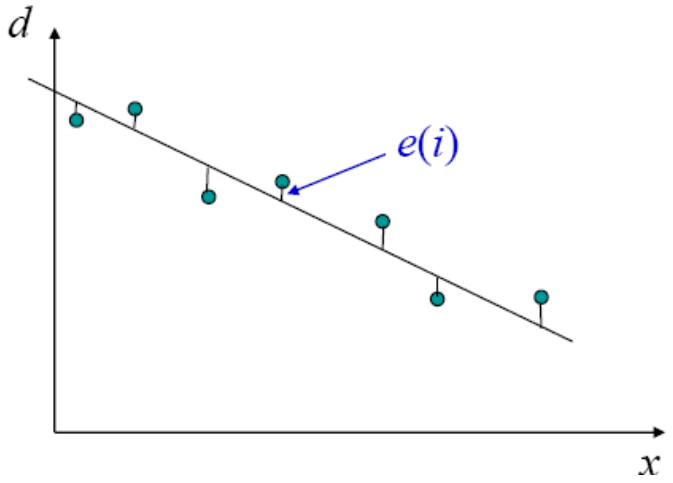
Does it also converge in finite steps? No.

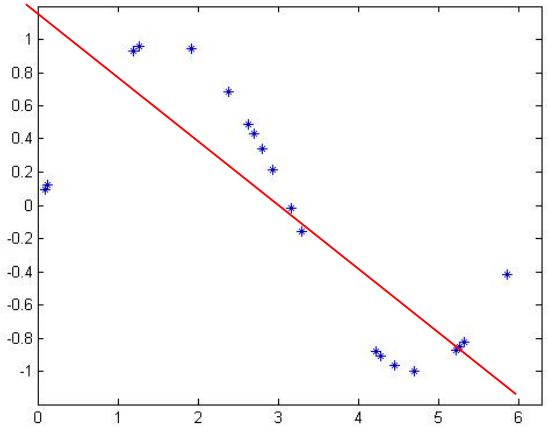
The fundamental limits of Single Layer Perceptrons

For Pattern Recognition Problem: Linearly Separable



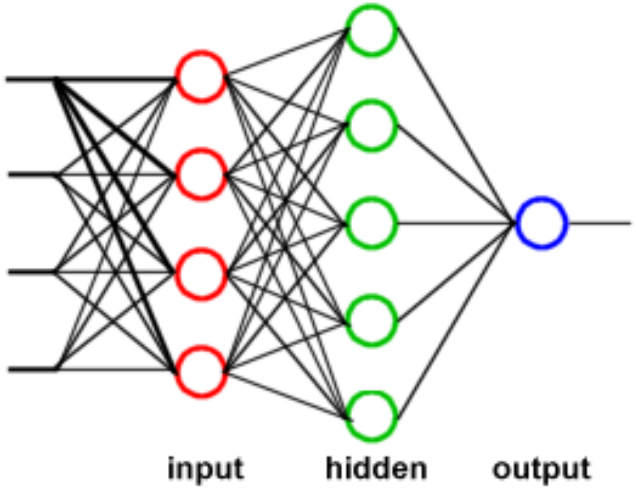
For Regression Problem: The process has to be close to a linear model!





David Rumelhart (1942-present)

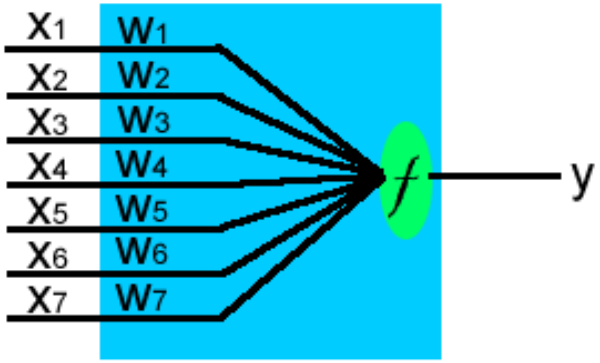
Multilayer Perceptron (MLP) and Back Propagation
David Rumelhart and his colleagues, 1986

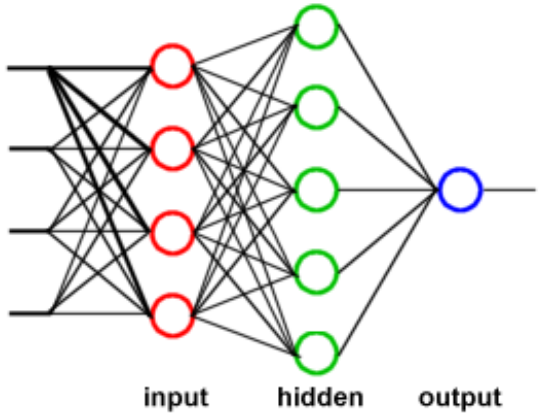


The hidden layer provides a nonlinear map from the original input space to a feature space.

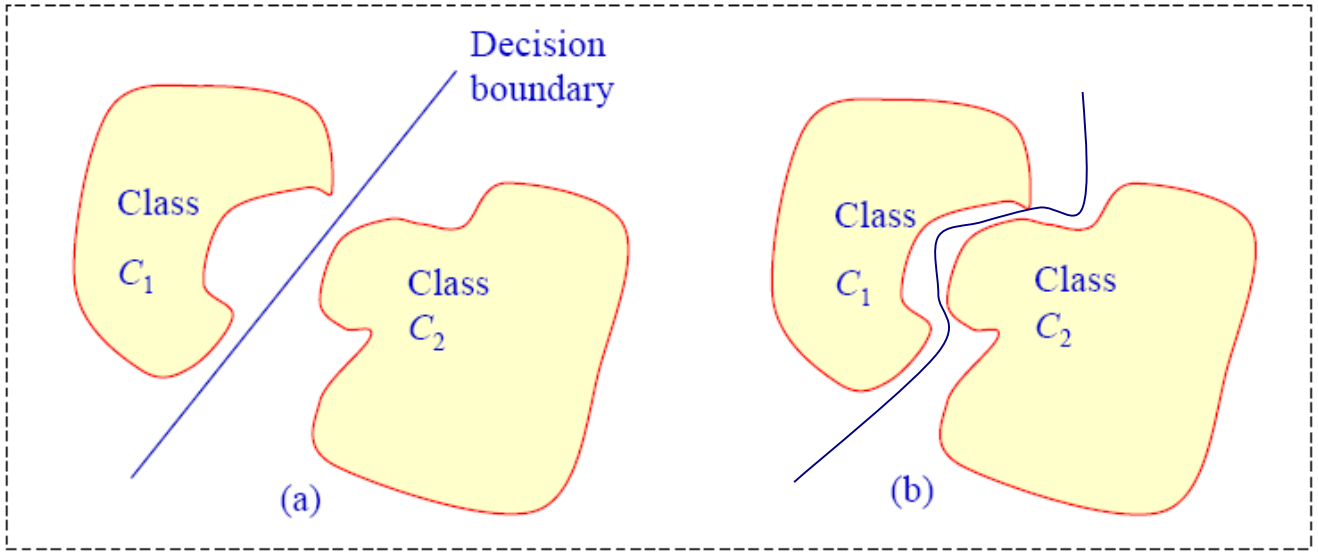
The nonlinearly separable problem in the input space can become linearly separable problem in the feature space!

Single Layer Perceptron v.s. Multi-layer Perceptrons



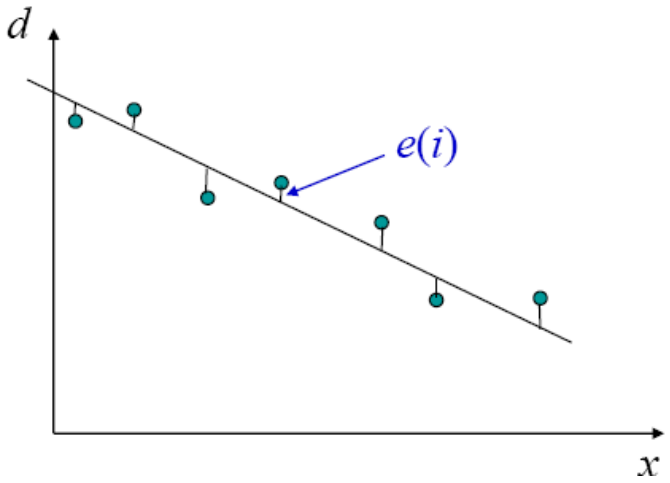


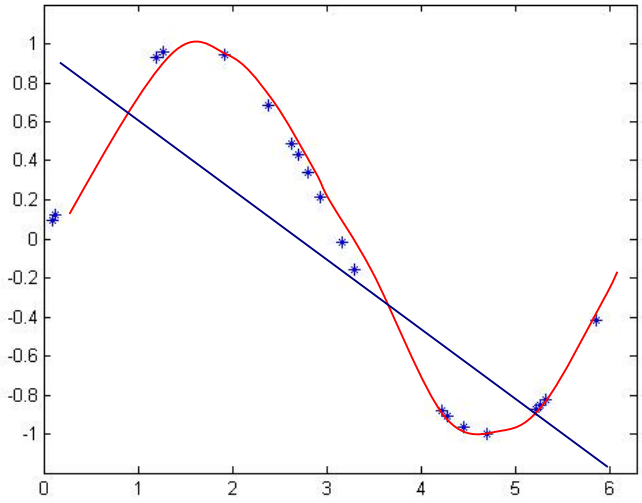
Pattern Recognition Problem:



Single Layer Perceptron v.s. Multi-layer Perceptrons

Regression Problem:





Multi-layer Perceptrons can approximate any bounded continuous functions!

The learning algorithms are based upon the steepest descent method:

$$w(k+1) = w(k) - \eta g(k)$$

$$w(n+1) = w(n) + \eta e(n) x(n)$$

Output Error

Input Signal

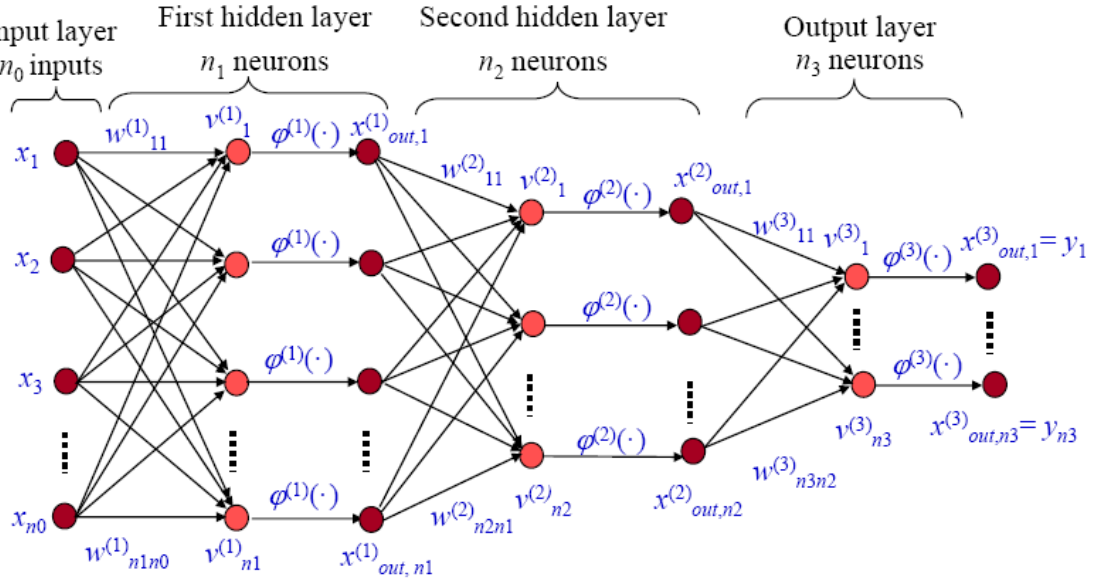
$$w_{ji}^{(s)}(k+1) = w_{ji}^{(s)}(k) + \eta^{(s)} \delta_j^{(s)} x_{out,i}^{(s-1)}$$

$\Delta w_{ji}^{(s)}$

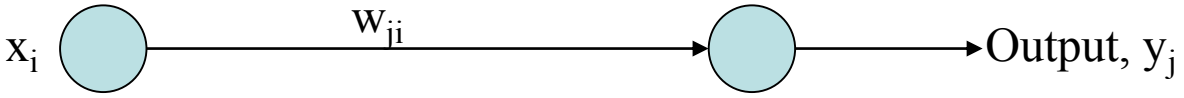
Output Error

Input Signal

The learning of the Multi-layer perceptron



$$w_{ji}^{(s)}(n+1) = w_{ji}^{(s)}(n) + \eta \delta_j^{(s)}(n) x_{out,i}^{(s-1)}(n)$$

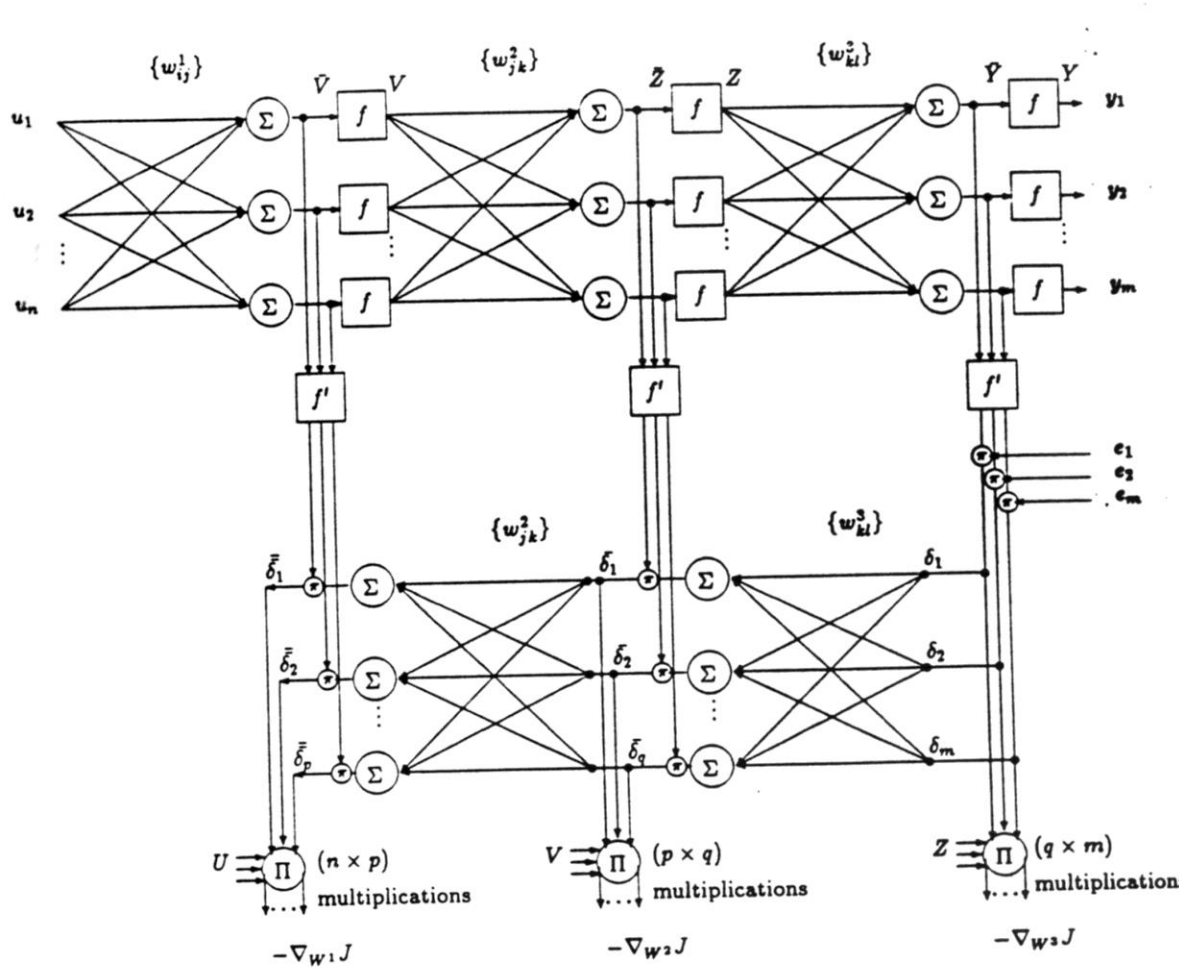


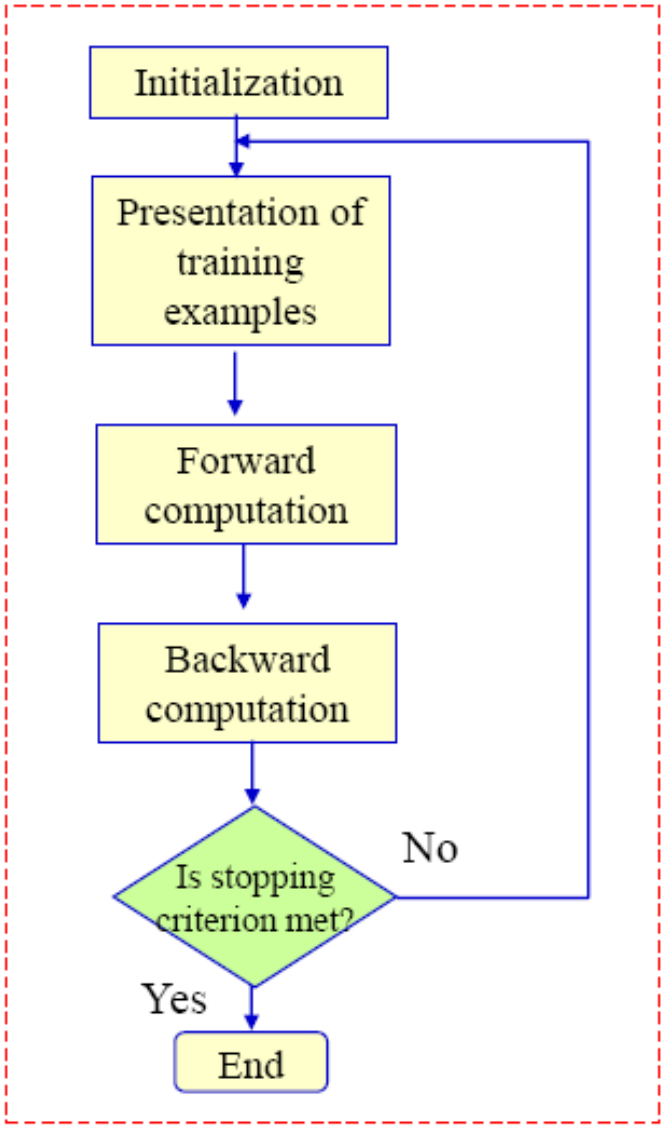
The adjustment of the synaptic weight only depends upon the information of the input neuron and the output neuron, and nothing else.

The output of the neuron depends upon all the connected neurons!

The adjustment of the synaptic weight is proportional to both the input signal and the output error.

Signal-flow graphic representation of BP





How to design and train the neural network?

How many hidden layers?

Normally one hidden layer is enough. Two hidden layers may be better if the target function can be clearly decomposed into sub-functions.

How many hidden neurons?

If the geometrical shape can be perceived, then use the minimal number of line segments (or hyper-planes) as the starting point.

For higher dimensional problem, start with a large network, then use SVD to determine the effective number of hidden neurons.

How to choose activation function in the hidden layers?

Hyperbolic tangent (tansig) is the preferred one in all the hidden neurons.

How to choose activation functions in the output neurons?

Logsig for pattern recogntion, purelin for regression problem.

How to pre-process the input data?

Normalize all the input variables to the same range such that the mean is close to zero.

When to use sequential leaning? And when to use batch learning ?

Batch learning with second order algorithm (such as trainlm in MATLAB) is usually faster, but prone to local minima. Sequential learning can produce better solution, in particular for large database with lots of redundant samples.

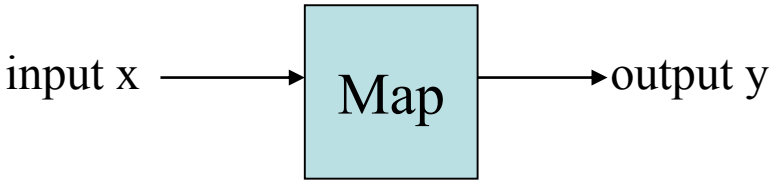
How to deal with over-fitting?

You either identify the minimal structure, or use regularization (trainbr in MATLAB).

What can MLP do?

MLP can solve regression and pattern recognition problems.
They all have the following characteristic:

There exists a function (map) between inputs x and outputs y: y=F(x)



Unfortunately, the mathematical form of this function (map) is unknown!

If the model is just too difficult to build,

or the map is too complicated to be expressed by any known simple functions,

then we can use MLP to approximate this function.

MLP is an universal approximator! It can approximate any bounded function!

All it needs is a training set:

Given a set of observations of input-output data:

{(x(1),d(1)),(x(2),d(2)), (x(3),d(3)),…,(x(N), d(N))}

Use this training data to train the MLP such that the difference between the desired outputs and the outputs of the MLP are minimized.

NETtalk --**Terrence Sejnowski and Charles Rosenberg in 1987.**

NETtalk was created to learn how to correctly pronounce English from written English text.

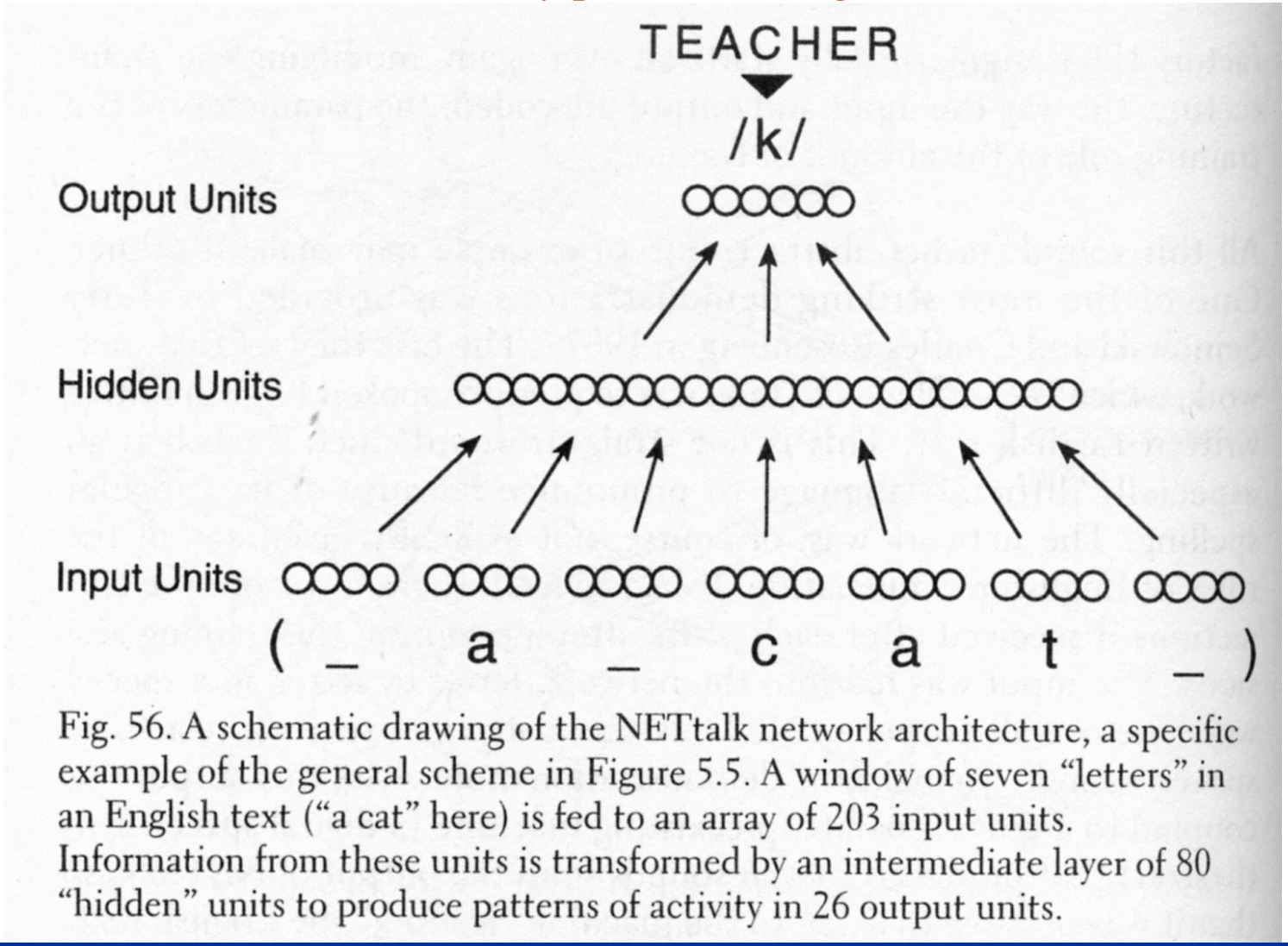
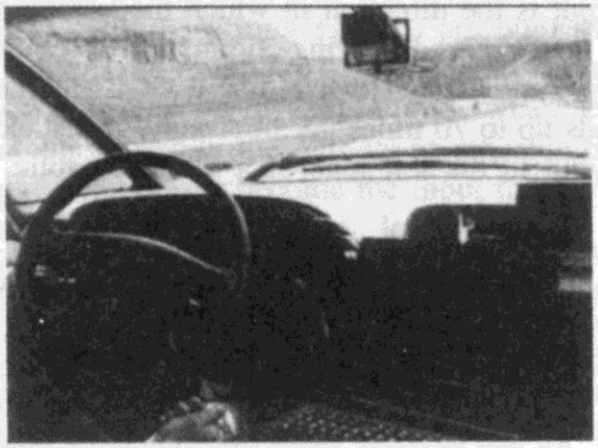
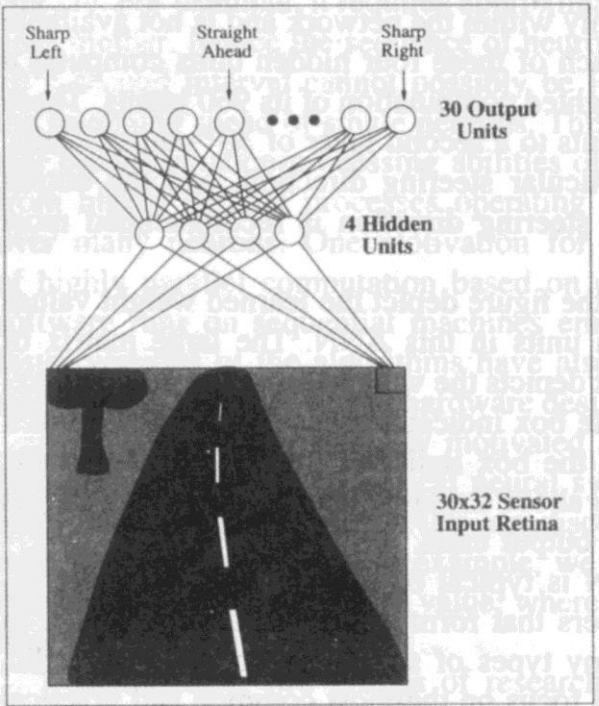
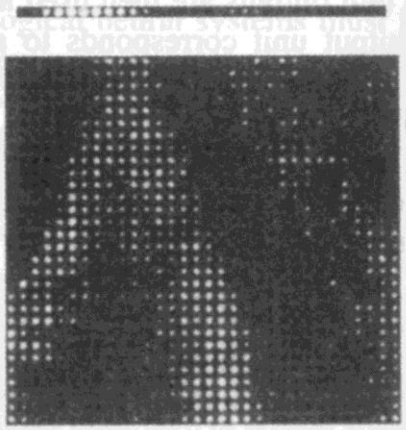


Fig. 56. A schematic drawing of the NETtalk network architecture, a specific example of the general scheme in Figure 5.5. A window of seven "letters" in an English text ("a cat" here) is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 "hidden" units to produce patterns of activity in 26 output units.







"Autonomous Land Vehicle In a Neural Network" (ALVINN)
by Dean Pomerleau and Todd Jochem, 1993, CMU

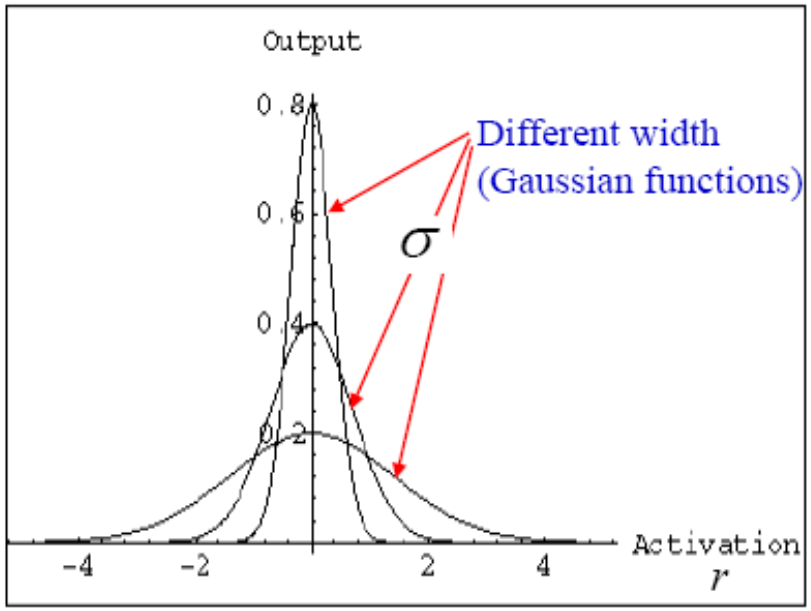
The state of the art self-driving car vehicle.

Radial Basis Functions (RBFs)

The activation of a hidden unit is determined by the distance between the input vector and a prototype vector, the center.

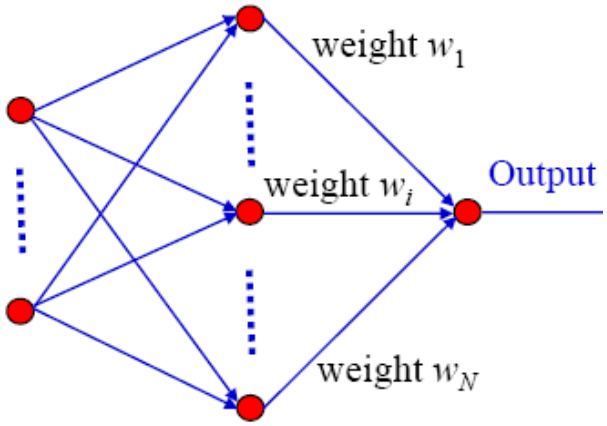
$$\varphi(x) = \varphi(\| x - c \|) = \varphi(r)$$

In most cases, if the distance is zero, the activation would be maximum. The activation level will drop off further away from the center.



$$y(x) = \sum_{i=1}^{M} w_i \varphi_i(\| x - \mu_i \|) + b$$

$$\varphi_i(\| x - \mu_i \|) = \exp(-\frac{\| x - \mu_i \|^2}{2\sigma_i^2})$$



We know that MLP can approximate any bounded continuous function. Can RBFN also do this?

RBFN can approximate any bounded continuous function!

It is just an existence result! It does not tell you how to obtain the weights.

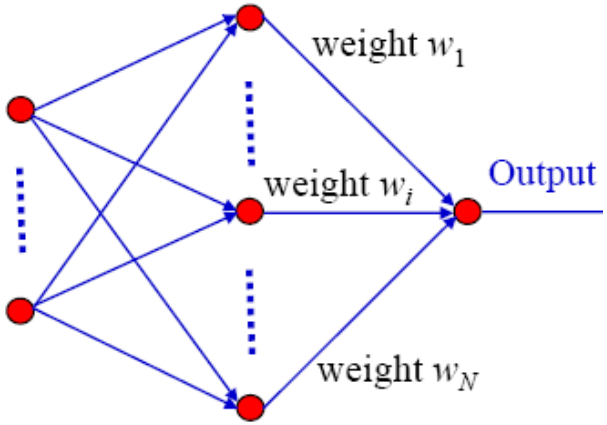
Given a set of sampling points and desired outputs {(x(i), d(i)), i=1,…,N}

How to find out the parameters $\{w_i\}$, $\{\mu_i\}$ and $\{\sigma_i\}$ such that the cost function

$$E(w) = \sum_{i=1}^{N} e(i)^2 = \sum_{i=1}^{N} (d(i) - y(i))^2$$

is minimized?

Hybrid training of RBF networks



Two stage 'hybrid' learning process:

(Stage 1) Parameterize hidden layer of RBFs.

- hidden unit number (M)
- centre/position (μ_i)
- spread/width (σ_i)

Use unsupervised methods as they are quick.

1. Random Selection
2. Clustering

(Stage 2) Find weight values between hidden and output units.

Aim: Minimize sum-of-squares error between actual outputs and desired responses.

$$E(w) = \sum_{i=1}^{N} e(i)^2 = \sum_{i=1}^{N} (y(i) - d(i))^2 = (\Phi w - d)^T (\Phi w - d)$$

$$\frac{\partial E(w)}{\partial w} = 0 \quad \Longrightarrow \quad w = (\Phi^T \Phi)^{-1} \Phi^T d$$

Supervised RBF Network Training

Supervised training of the basis function parameters will generally give better results than unsupervised procedures, but the computational costs are usually enormous.

The obvious approach is to perform gradient descent on a sum squared output error function as we did for MLPs. The error function could be,

$$E = \frac{1}{2}\sum_{j=1}^{N}\left(e_j\right)^2$$

$$\varphi_i(\cdot) = \exp\left(-\frac{\left\|\boldsymbol{x}_j - \boldsymbol{\mu}_i\right\|^2}{2\sigma_i^2}\right)$$

$$e_j = d_j - y(\boldsymbol{x}_j) = d_j - \sum_{i=0}^{M} w_i \varphi_i(\boldsymbol{x}_j, \boldsymbol{\mu}_i, \sigma_i)$$

and we would iteratively update the weights/basis function parameters using

$$\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n) = -\eta \cdot \boldsymbol{g}(n)$$

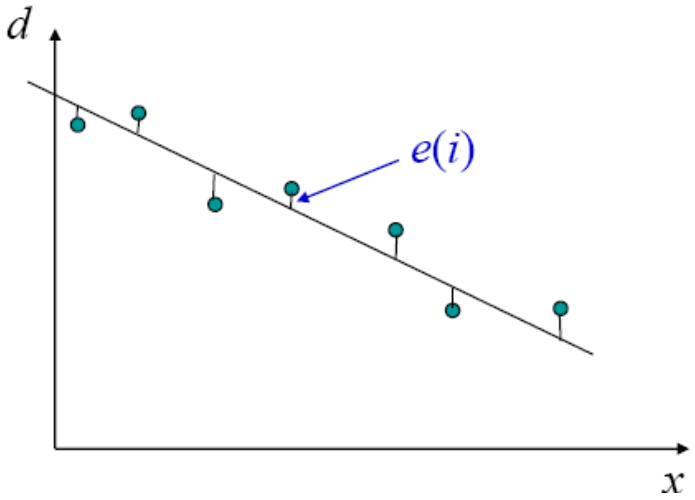
$$\Delta w_i = -\eta_w \frac{\partial E}{\partial w_i} \qquad \Delta \boldsymbol{\mu}_i = -\eta_\mu \frac{\partial E}{\partial \boldsymbol{\mu}_i} \qquad \Delta \sigma_i = -\eta_\sigma \frac{\partial E}{\partial \sigma_i}$$

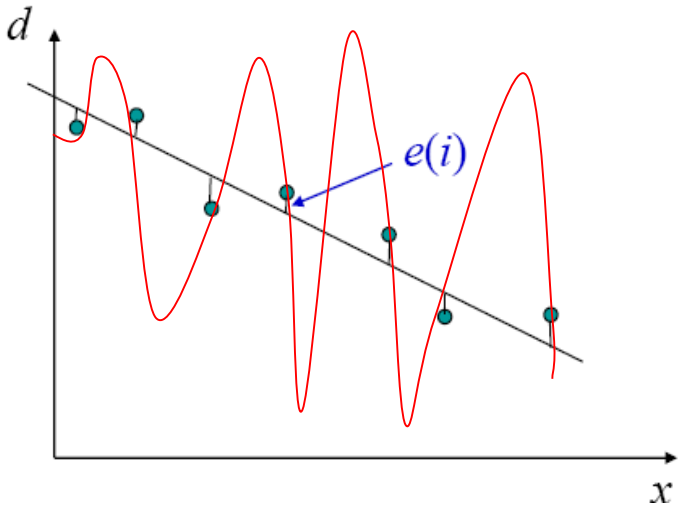
We have all the problems of choosing the learning rates η, avoiding local minima and so on, that we had for training MLPs by gradient descent.

Second order methods can also be attempted.

Regularization Theory

Both MLP and RBFN can lead to to over-fitting (learn the noise present in the training data) and result in poor generalization.



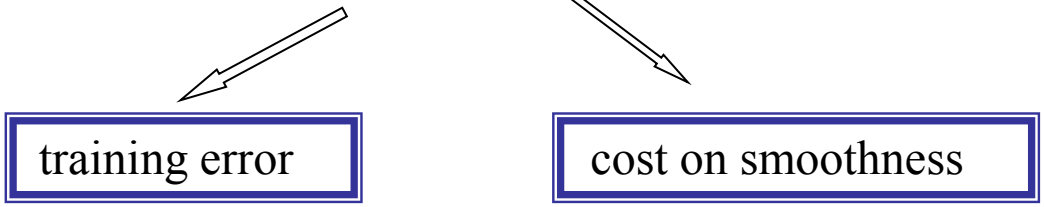


An alternative approach to cope with over-fitting comes from the theory of regularization, which is a method of controlling the *smoothness* of mapping functions.

Regularization Methods

Cost function:

$$F = E_D + \lambda E_w$$



It involves adding an extra term to the error measure which penalizes mappings that are not smooth.

For RBFN, a simple way to choose the penalty term is the size of the weights.

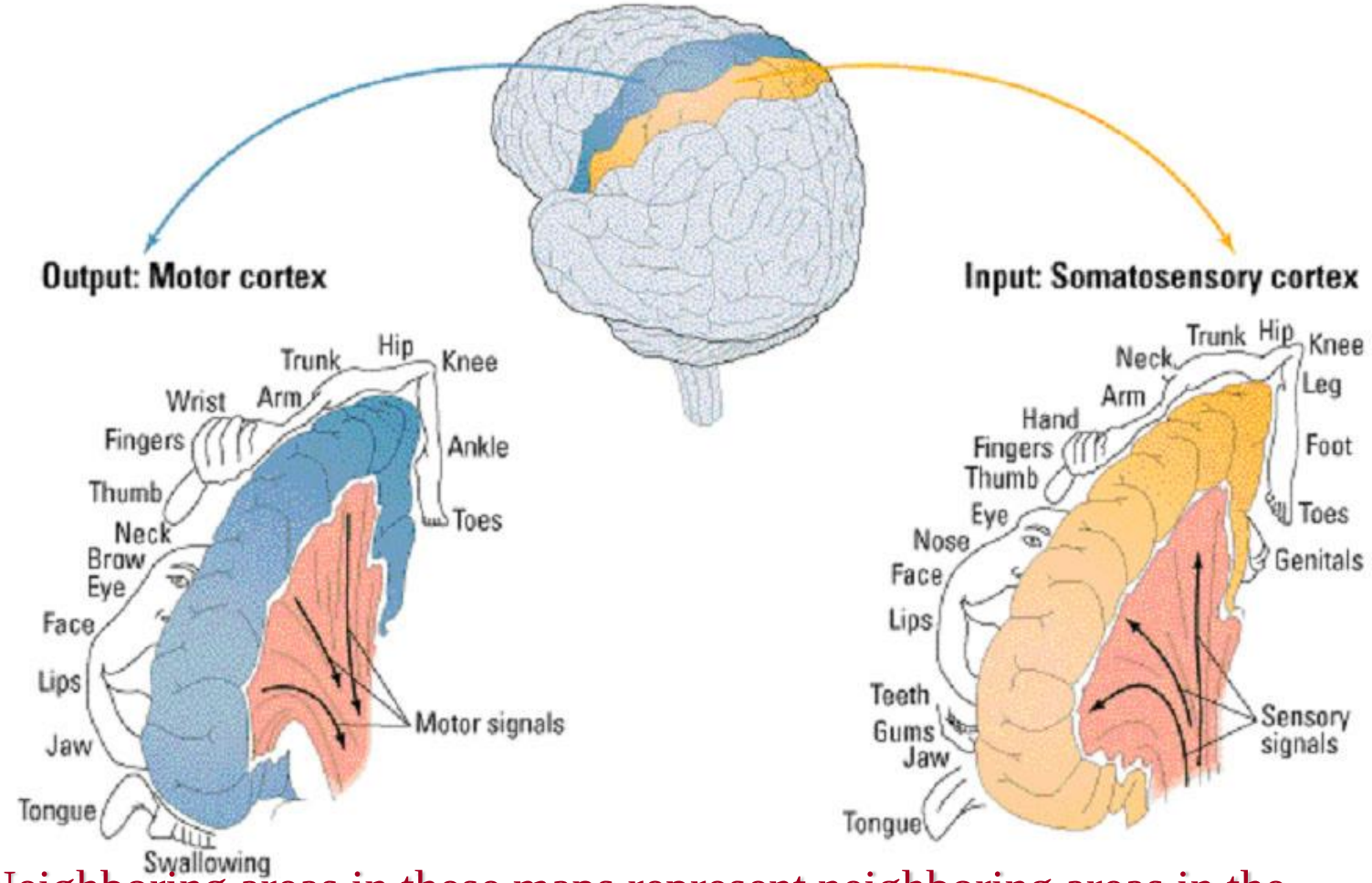
$$F(w) = \frac{1}{2}\sum_{i=1}^{N}(y(i)-d(i))^2 + \frac{1}{2}\lambda \| w\|^2 = \frac{1}{2}(\Phi w - d)^T(\Phi w - d) + \frac{1}{2}\lambda w^T w$$

$$w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T d$$

λ is the regularization factor

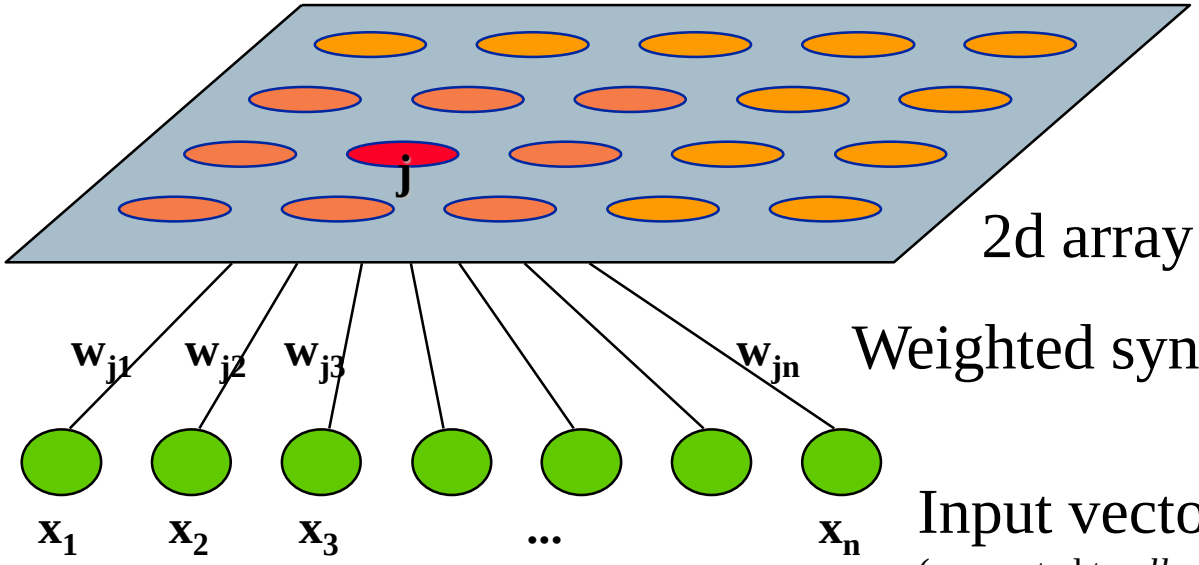
For MLP, regularization algorithm is more involved. Trainbr in MATLAB

Feature Mapping of the Human Cortex



Neighboring areas in these maps represent neighboring areas in the sensory input space or motory output space.

SOM – Architecture and Algorithm Overview



2d array of neurons

Weighted synapses

Input vector
(connected to *all* neurons in lattice)

1. **Randomly initialise all weights**
2. **Select input vector $x = [x_1, x_2, x_3, \ldots, x_n]$.**
3. **Compare x with weights w_j for each neuron j to determine winner.**
4. **Update winner so that it becomes more like x, together with the winner's *neighbours*.**
5. **Adjust parameters: learning rate & 'neighbourhood function'.**
6. **Repeat from (2) until the map has converged (i.e. no noticeable changes in the weights) or pre-defined no. of training cycles have passed.**

What lies in the future for AI or computational intelligence?

What is happening now?

What would happen in the distant future?

It could be better or worse!

And people are worried!